28 FEB 2003

G00002

166166

SCHEDULING RESOURCE REQUESTS IN A COMPUTER SYSTEM

Background

The invention relates to deterministic scheduling of requests in systems.

Software layers in a system (such as a computer) typically include an operating system and application programs. When application programs are executed, one or more processes, tasks, or other basic units of work or execution entities may be created. With certain operating systems, such as the Windows 95 or Windows NT® operating systems from Microsoft Corporation, each process may contain one or more units of work (referred to as threads) that execute code contained in the process' address space to perform assigned functions. Threads belonging to a parent process may be assigned to perform different functions. For example, with a spreadsheet process, threads may be created to calculate, print, accept user inputs, provide help functions, and so forth. With other operating systems, a task or process may constitute the basic unit of work or execution entity that may be scheduled for execution by a central processing unit (CPU).

Operating systems may include a scheduler to manage multiple active threads or processes. Different types of operating systems may have different scheduling schemes. For example, with some Windows operating systems, time slices are assigned to active threads in a round robin scheme during which corresponding threads are allowed to execute. Further, with some Windows operating systems, priority classes may be assigned to the threads. Threads in the highest priority class are first executed in their assigned time slices followed by threads in lower priority classes. Thus, in any particular priority class, the scheduling may be performed in a round robin fashion. A thread continues to run until one or more events occur: the time slice finishes or the thread is preempted by another thread of a higher priority class that is ready to run.

Different processes, threads, or other units of work may use system resources differently. For example, in a video playback and decode process, data may be transferred from a compact disc (CD) or digital video disc (DVD) drive to system memory, transferred between the CPU and system memory, and transferred from the system memory to video memory for display by a graphics card on a monitor. Generally, data transferred from the CD or DVD drive to system memory is slower than data transferred between the graphics card and system memory, which in turn may be slower than the transfer of data between the CPU and system memory. Thus system resources (including, for example, system memory, buses,

and other devices) may be used differently depending on the requirements of the different units of work or execution entities.

Conventional operating systems typically do not effectively account for the different resource requirements of different processes, threads or other units of work. Such conventional operating systems schedule processes, threads, or other units of work at the application level; for example, the processes, threads or other units of work associated with each application are assigned to a predetermined priority class. Typically, requests from the units of work are scheduled according to the preassigned priority and scheduling protocol without regard to whether needed system resources are available.

Summary

In general, according to one embodiment, a system includes a resource, execution entities adapted to issue requests for the resource, and a storage location containing slot assignments associated with requests for the resource from each execution entity. A controller is coupled to the resource and adapted to access the storage location and to process requests for the resource from the execution entities according to the slot assignments.

Other features will become apparent from the following description and from the claims.

Brief Description Of The Drawings

Fig. 1 is a block diagram of a system of an embodiment of the invention.

Figs. 2A and 2B are a block diagram of layers in the system of Fig. 1.

Fig. 3 illustrates a scheduling cycle according to an embodiment.

Fig. 4 is a flow diagram of a scheduling module according to an embodiment in the system of Fig. 1.

Fig. 5 is a flow diagram of a basic input/output system (BIOS) routine according to an embodiment in the system of Fig. 1.

Fig. 6 is a flow diagram of an operating system according to an embodiment in the system of Fig. 1.

Detailed Description

A system according to embodiments of the invention includes various resources that are utilized or accessed by software of firmware layers or modules running in the system. As examples, system resources may include system memory, one or more buses, and other devices. A scheduler according to embodiments running in the system schedules requests from basic units of work or execution entities (e.g., processes, tasks or threads) created by the software and firmware layers according to predetermined criteria, which may in some embodiments include whether certain system resources are available and the latency and bandwidth requirements of the requests. According to some embodiments, the scheduler is provided feedback of system resource availability and utilization. By utilizing a deterministic approach in which availability and utilization of requested resources may be determined by a scheduler, the scheduler is better able to guarantee that a request from an execution entity is serviced according to the bandwidth and latency needs of the requesting entity.

Referring to Fig. 1, a block diagram is shown of a system 10, which may be, for example, a general-purpose or special-purpose computer, other microprocessor- or microcontroller-based system, a hand-held computing device, a set-top box, an appliance, a game system, or any other system including a control device such as an application specific integrated circuit (ASIC) or a programmable gate array (PGA).

Although this description makes reference to specific configurations and architectures of the various layers of the system 10, it is contemplated that numerous modifications and variations of the described and illustrated embodiments may be possible.

In the embodiment of Fig. 1, the system 10 includes a central processing unit (CPU) 100 that is coupled through a host bridge controller 102 that may include a memory controller 103 coupled to main memory 104 and a graphics interface 105 coupled to a graphics controller 106. The graphics interface 105 may be, for example, an Accelerated Graphics Port (A.G.P.) interface according to the Accelerated Graphics Port Interface Specification, Revision 2.0, published in May 1998. The host bridge controller 102 may also include a cache controller 107 to control a second level (L2) cache memory 109. The host bridge controller 102 includes a bus interface 111 coupled to a system bus 112, which in one embodiment may be the Peripheral Component Interconnect (PCI) bus that operates according to the PCI Local Bus Specification, Production Version, Revision 2.1, published in June 1995, or in alternative embodiments other types of interface protocols.

For example, in another configuration, the host and bridge controllers may be replaced with a memory hub and an input/output hub coupled by a link. In such a configuration, memory and graphics interface circuitry may be in the memory hub and bridge controllers may be in the I/O hub.

The system bus 112 may be coupled to a storage device controller 114 that controls access to one or more storage devices such as a hard disk drive 115 or a compact disc (CD) or digital video disc (DVD) drive 116. Other devices may also be coupled to the system bus 112, such as a network interface card and slots coupled to peripheral devices (not shown). The system according to various other integration levels may have controllers implemented in different blocks. For example, the hard disk drive and CD or DVD drive controller may be included in the system bridge 110.

The system 10 may also include a secondary or expansion bus 120. A system bridge controller 110 is coupled between the system bus 112 and the expansion bus 120. The system bridge controller 110 may include a system bus interface 113 coupled to the system bus 112 and an expansion bus interface 119 coupled to the expansion bus 120. The system bridge controller may also include a Universal Serial Bus (USB) interface 117 that is coupled to a USB port 118, as described in the Universal Bus Specification, Revision 1.0, published in January 1996. The expansion bus 120 may be coupled to various peripheral devices 122 and a non-volatile memory 124. The components and devices coupled to the buses, the buses themselves, and the system memory may form part of the system resources that are utilized or accessed by requests from units of work or execution entities in the system 10.

Referring to Figs. 2A-2B, various software and hardware layers in the system 10 are illustrated in more detail. As examples, the system 10 may include an operating system (OS) 220 and processes 222 and 224. In the ensuing description, it is assumed that the operating system 220 is a thread-based system, such as some Windows operating systems, in which each process may include one or more threads. It is to be understood, however, that the request scheduling scheme according to the described embodiments may be implemented in operating systems with differently configured execution entities or units of work.

As illustrated in Fig. 2A, threads 228 and 229 belong to the process 222 and threads 230 and 231 belong to the process 224. The threads may communicate with the OS 220 through a predefined interface, such as an application programmable interface (API) that may be defined under the operating system. Alternatively, a "third party" API may be used. The

OS 220 includes a scheduler 232 that schedules requests from the active threads through the predefined interface. In one embodiment, the scheduler 232 may be associated with a device driver 240 that is capable of accessing memory, I/O, or other defined locations in the system 10 to communicate with hardware components to perform scheduling acts according to embodiments of the invention. The elements 232 and 240 may collectively also be referred to as a scheduler. In further embodiments, the scheduler may be separated into more modules or layers.

Upon receiving a request from a thread, the scheduler 232 schedules the request based on feedback communication from hardware components, the number of outstanding requests, and the latency and bandwidth requirements of the requesting thread.

According to one embodiment, the scheduler 232 stores requests into a request queue 204 having a predetermined number of entries. Each entry in the request queue 204 may be associated with a status flag in a status field 206 that indicates if a particular request has been processed.

A set of tables or table segments 202 stored in system memory 104 (or in some other suitable storage location), which is accessible by the scheduler 232 through the device driver 240, identifies channel assignments for corresponding active threads, each having a thread identifier (ID). Each channel is defined as having a predefined number of cycles of a basic clock, such as a clock generated by a clock generator 250. The number of basic clocks per channel is selected based on a desired granularity. Each of the table or table segments 202 corresponds to a resource in the system 10, including for example the system memory 104, the graphics card 106, the system bus 112, the expansion bus 120, the USB port 118, and so forth.

According to an embodiment, the bridge controllers 102, 110 also store tables or table segments corresponding to the various system resources that keep track of channel assignments for specific threads by thread IDs. The tables or table segments in the bridge controllers are loaded based on the corresponding tables 202 maintained by the OS 220. In the illustrated embodiment, tables or table segments 302A, 302B, and 302C may be stored by the host bridge controller 102 to keep track of channel assignments for the system memory 104, the system bus 112, and the graphics card 106, respectively. The system bridge controller 110 may store tables or table segments 302D and 302E to keep track of channel

assignments for the USB port 118 and the expansion bus 120. Other tables may also be maintained for other system resources.

Although illustrated as being stored in the bridge controllers 102, 110, the tables 302A-302E may be stored in other suitable locations, such as system memory 104 or external storage devices. Alternatively, the different system resources may be controlled by controllers distributed throughout the system rather than integrated in the bridge controllers 102, 110 as illustrated.

The tables 302A-302E may be updated periodically by the OS 220 as channel assignments change. In each of the tables 302A-302E, the threads may be assigned the same or different numbers of channels. For example, a thread associated with a first thread ID may be assigned a first number of channels, a thread associated with a second thread ID may be assigned a different number of channels, and so forth.

The channels assigned to the threads in the tables 302A-302E define thread request execution windows or slots within an overall scheduling cycle 400, as illustrated in Fig. 3. The scheduling cycle 400 includes multiple thread request execution windows or slots 402_0, 402_1, ..., 402_{N-1}, and 402_N, each including an assigned number of channels. Each request execution windows 402 is assigned to execution of requests from a thread.

In one embodiment, the first window 402_0 is assigned to the scheduler 232 for maintaining coherency between the OS tables 202 and corresponding tables 302 in the bridge controllers 102, 110. The remaining windows 402_1 to 402_N may be assigned to requests from various other threads. In the illustrated embodiment, the tables 302A-302E are updated by the scheduler device driver 240 once every scheduling cycle 400 in the coherency window 402_0. During the time period of the coherency window 402_0, or alternatively, in another window 402_i, the scheduler device driver 240 may also read contents of status registers 304A-304E in the bridge controllers 102, 110 to determine which requests have completed. The scheduler 232 is thus provided feedback of which requests have been completed and which are still pending to allow it to keep track of which system resources are available. In this manner, when a request from a thread is received by the scheduler 232, the scheduler 232 can determine whether sufficient resources are available to process the request.

Referring again to Figs. 2A-2B, according to an embodiment, each of the bridge controllers 102, 110 contain various queues to store requests for various resources in the system 10. By way of example, the host bridge controller 102 includes a memory queue 310

in the memory controller 103. Requests received by the memory controller 103 from various sources in the system 10, including the CPU 100 through a CPU bus interface 312 and devices on the system bus 112 through the system bus interface 111, are stored in the memory queue 310 for execution. In addition to the memory address and data information, the memory queue 310 is also adapted to store the associated thread ID of a memory request.

In some embodiments of the invention, the CPU 100 fetches an instruction along with the associated thread ID. The thread ID is passed along and stored in queues of the bridge controllers 102, 110.

A scheduler controller 314 in the host bridge controller 102 receives output values from a counter 306 and the tables or table segments 302A-302C. The counter 306, which may be clocked by the basic clock from the clock generator 250, may be adapted to count through the channels in the scheduling cycle 400. From the counter 306 value, the scheduler controller 314 is able to determine the current thread request execution window 400_i (i=0 to N) within the scheduling cycle 400. Based on which window 400_i is active and the channel assignments in the table 302A, a request associated with the corresponding thread ID in the memory queue 310 may be selected for processing by the memory controller 103.

The selected request is executed within the current window 402_i. Upon completion of the request, the host bridge controller 102 may return a request completed status by programming appropriate bits in the status register 304A. In the next coherency window 402_0, the CPU 100 under control of the scheduler device driver 240 reads the status register 304A (as well as the other status registers 304B-304E) to determine which requests have completed. The scheduler device driver 240 then updates flags 206 of completed requests in the request queue 204.

In addition to the memory queue 310, the host bridge controller 102 in one embodiment may also include a system bus queue 316 in the system bus interface 111 and a graphics card request queue 318 in the graphics interface 105. Requests for the system bus 112 are entered into the system bus queue 316 while requests for the graphics card 106 are entered into the graphics card queue 318. The requests are associated with thread IDs so that the scheduler controller 314 can select the appropriate request for processing by the bus interface 111 or the graphics interface 105 based on the current thread request execution window 402_i and the channel assignments stored in the tables 302B and 302C. Completion

of requests in the queues 316 and 318 are indicated by status registers 304B and 304C, respectively.

Similarly, the system bridge controller 110 includes an expansion bus queue 320 to store requests targeted for the expansion bus 120 and a USB bus queue 322 to store requests targeted for the USB port 118. A counter 308, which may be clocked by the basic clock from the clock generator 250, counts through the channels of the scheduling cycle 400. Based on the current thread request execution windows 402_i as indicated by the counter 308, and on the channel assignments stored in the tables 302D and 302E, a scheduler controller 324 in the system bridge controller 110 determines which of the requests in the queues 320 and 322, respectively, are to be processed. Completion of the requests are indicated by status registers 304D and 304E.

Referring to Fig. 4, the scheduler device driver 240 that works in cooperation with the scheduler 232 waits for receipt of certain events (at 502). If a request from a thread is received, which may be in the form of an application programmable interface (API) call, the scheduler device driver 240 accesses (at 504) the request queue 204 and channel assignment tables 202 so that the scheduler 232 can determine (at 506) if resources are available to process the thread request.

If the scheduler 232 determines that resources are not available to adequately process the request, then the requesting thread is notified (at 508). In response to the notification, the thread can wait for some period of time before reissuing the request or the thread may otherwise gracefully handle the situation. If the requested resources are available, then the request from the thread is added (at 510) to the request queue 204.

When a particular request in the schedule queue 204 has been processed and completed, a complete flag in the flag field 206 of the queue 204 may be set, as discussed above.

The scheduler device driver 240 itself is a thread capable of issuing requests, such as to access memory locations corresponding to the request queue and the thread channel assignment tables. Requests from the schedule device driver thread entered into the request queue 204 (at 520) may be processed in the first window 402_0 of the scheduling cycle 400. Alternatively, another window 402_i may be assigned to the scheduler device driver 240.

During the coherency window 402_0, the CPU 100 under control of the scheduler device driver 240 updates (at 522) the bridge controller tables 302A-302E as necessary to

change channel assignments. The CPU 100 may also access (at 524) the status register 304A-304E to determine which requests have completed. Alternatively, the table update and status register read acts may be separated. The CPU 100 under control of the device driver 240 may then update (at 526) the request queue 204.

The scheduler 232 can determine what system resources are needed by a request by looking at the request itself and the request parameters, such as parameters in an API call. For example, a parameter may specify access to a location in memory address space in the system memory 104. Another parameter may specify a location in I/O address space, which may be located in one of the buses 112, 120, in the graphics card 106, on the USB bus coupled to the USB port 118, or another location in the system 10. Based on the requested resources, the requests already in the queue 204 and channel assignments specified in the tables or table segments 202 as retrieved by the scheduler device driver 240, the scheduler device driver 232 can determine if a request may be processed in some reasonable manner. This may be defined according to criteria preprogrammed into the system 10 and loaded by a startup routine (e.g., a basic input/output system or BIOS routine). The scheduler 232 is aware of the latency and bandwidth requirements of the various threads in the system. The latency and bandwidth requirements may be used by the scheduler 232 to determine if sufficient resources are available to satisfy a thread request.

As an example, a thread may issue a request to transfer a frame of video data from video memory in the graphics card 106 to system memory 104. Given a frame size (e.g., 720 x 480 pixels) and a number of bits defined for each pixel, the bandwidth requirement of the video transfer may be determined based on a specified period of time in which the transfer has to occur. In addition, the latency of the transfer request may also be known. Based on bandwidth and latency information, and based on the outstanding requests for resources that are outstanding, the scheduler can determine if the video transfer request can be processed by available resources. If not, the thread is informed by the scheduler, and the thread may respond in one of a number of ways, including waiting before issuing another request or breaking a request into several parts.

Referring to Fig. 5, system initialization is performed by a system BIOS routine according to an embodiment. After system reset has placed system hardware into an initial state, the CPU 100 starts to execute instructions in the power-on self test (POST) procedure of the BIOS, which is responsible for initializing components in the system 10 to known

states and for constructing system configuration information for the OS 220 to use. After some initialization tasks are performed (at 602) in the system 10, the BIOS routine next sets up (at 604) the system memory 104 or other suitable storage location to store the channel assignment tables or table segments 202. The BIOS routine may specify the number of basic clocks for each channel (e.g., one or multiple clocks per channel) and the total width of the scheduling cycle 400. Specific memory addresses may be reserved to store the tables or table segments 202. In addition, criteria may be specified for use by the scheduler 232 in determining whether to accept or reject a thread request.

Next, default channel assignments may be loaded (at 606) into the tables. For example, certain threads associated with the OS (such as the scheduler and other system management layers) may be assigned to windows in the cycle 400. In addition, the BIOS may set up default windows having predetermined numbers of channels to handle thread requests that have not been assigned to specific windows. The BIOS can poll the configuration space of the system to determine the type of processor available, if the processor is a multi-processing system, and other information. From the information, the BIOS can determine the capabilities of the system. Based on such determined capabilities, the BIOS can assign the number of channels in the default windows accordingly. Next, system components are initialized and configured by the BIOS routine (at 608). The OS 220 is then booted (at 610).

Referring to Fig. 6, after the OS 220 is booted, it first identifies (at 650) the number of active threads in the system 10. The OS 220 queries (at 652) each thread for its bandwidth and latency requirements for resources in the system 10. Some threads are aware of what their latency and bandwidth requirements are for each system resource. For example, a thread associated with a multimedia process may have "real time" requirements that can tolerate a relatively small latency for data transfers and that require high data transfer throughput. Other threads may be able to tolerate higher latencies and lower data transfer bandwidths. Based on a comparison of the different latency and bandwidth requirements from the active threads, the number of channels to assign the different request windows 402_i corresponding to the different threads may be set by the OS (at 654) to favor the threads requiring lower latencies and higher bandwidths. As a result, those types of threads may be assigned a larger number of channels and possibly multiple windows 402_i. The multiple assigned windows 402_i may be consecutive or dispersed in the scheduling cycle 400. If a

thread does not provide the latency and bandwidth information for any system resource, that thread may be assigned to default windows 402_i in the scheduling cycle 400 of the tables or table segments 202.

Based on the calculated number of channels, the tables or table segments 202 may be loaded (at 656) with the channel assignments according to thread IDs.

Thus, according to some embodiments, a scheduling scheme schedules requests from threads in the system by determining if data flow requirements may be satisfied based on availability of resources and channel assignments. The availability of resources is indicated by hardware components to a scheduler in the system. Further, based on the type of request and request parameters, the scheduler is able to determine what resources are needed by the particular request.

Other embodiments are within the scope of the following claims. For example, with different operating systems, the basic units of work or execution entities in the system may not be threads but may be processes or other defined units. Further, the hardware components in the system may be differently configured. The acts performed by the software and firmware illustrated modules and layers may be varied.

While the invention has been disclosed with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of the invention.

What is claimed is:

What is claimed:

1. A system comprising:

a resource;

execution entities adapted to issue requests for the resource;

a storage location containing slot assignments associated with outstanding requests for the resource from each execution entity; and

a controller operatively coupled to the resource and adapted to access the storage location and to process requests for the resource from the execution entities according to the slot assignments,

the controller to also determine if the resource is available for a given request based on the slot assignments associated with outstanding requests and on bandwidth and latency requirements of the given request.

2. The system of claim 1, further comprising a scheduler adapted to update slot assignments in the storage location.

3. The system of claim 2, further comprising a second storage location accessible by the scheduler and containing slot assignments for the execution entities.

4. The system of claim 1, wherein the given request includes an application programmable interface call associated with one or more parameters.

5. The system of claim 2, wherein the scheduler is adapted to access the controller to determine if a request has been processed.

6. The system of claim 1, further comprising a counter accessible by the controller to determine which slot is active.

7. The system of claim 6, wherein each slot includes one or more channels, the counter adapted to count through the channels.

8. The system of claim 1, wherein multiple slots are defined in a scheduling cycle, and the controller is adapted to process requests from the execution entities in the corresponding slots.

9. The system of claim 1, further comprising a device containing a queue to store requests of the execution entities and corresponding identifiers of the execution entities,

the device to control access to the resource.

10. The system of claim 9, the controller to further access the queue to schedule the first request.

11. A system comprising:

a system resource;

execution units;

an operating system comprising a scheduler to negotiate with the execution units to determine resource usage requirements; and

a storage location containing a data structure having identifiers of the execution entities and corresponding channel assignments, the scheduler to update the channel assignments in the data structure based on the resource usage requirements of an execution unit capable of accessing the system resource during its assigned channels,

the data structure further containing indications for indicating that a hardware component has completed execution of a request.

12. The system of claim 11, wherein the execution units are adapted to communicate bandwidth information for the resource to the operating system.

13. The system of claim 12, wherein the execution units are adapted to further communicate latency information for the resource to the operating system.

14. The system of claim 11, further comprising a device containing a queue to store requests of the execution entities and corresponding identifiers of the execution entities,

the device to control access to the system resource.

15. The system of claim 14, the scheduler to further access the queue to schedule the requests from the execution entities.

16. A method of scheduling requests from execution units in a system, comprising:

determining data flow information of the execution units for a system resource, the data flow information comprising at least one of bandwidth information and latency information relating to use of the system resource by the execution units;

storing a data structure to assign plural time windows to plural execution units, each time window comprising plural time slots during which a corresponding execution unit is given access to the system resource based on the data flow information;

programming a controller based on the data structure containing the plural time windows assigned to plural execution units; and

the controller processing requests for the system resource based on which time window is currently active, the data structure and the data flow information,

wherein the controller comprises an operating system scheduler.

17. The method of claim 16, further comprising determining if a request from a first execution unit can be processed based on the data structure and availability of the system resource.

18. The method of claim 17, further comprising determining availability of the system resource based on the data flow information and pending requests for the system resource.

19. The method of claim 16, further comprising storing a queue in a device to store requests of the execution entities and corresponding identifiers of the execution entities,

the device controlling access to the system resource.

20. The method of claim 19, further comprising the scheduler accessing the queue to schedule the requests.

21. An article including a storage medium containing instructions for scheduling requests from execution entities, the instructions causing a processor to:

receive a first request from an execution entity that includes an access to a system resource;

access a data structure containing time window assignments to assign plural time windows to plural execution entities;

determine if the system resource is available for the first request; and

schedule the first request from the execution entity based on accessing the data structure and determining if the system resource is available,

wherein determining if the system resource is available is based in part on an indication that a hardware component has completed execution of a request.

22. The article of claim 24, containing instructions for causing the processor to further notify the execution entity if the request cannot be processed because the resource is unavailable.