SEMAPHORE SYSTEM BASED ON PROCESS EVENTS

COPYRIGHT NOTICE

[0001] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever. The following notice applies to the software and data as described below and in the drawings hereto: Copyright © 2001, Intel Corporation, All Rights Reserved.

FIELD OF THE INVENTION

[0002] The invention relates generally to parallel processes, and more particularly, to semaphores for managing access to shared system resources.

BACKGROUND OF THE INVENTION

[0003] Systems that perform parallel processing comprise simultaneously executing processes. A common problem when using parallel processing is protecting the contents of a common resource.

[0004] One known way to implement this is to use a semaphore, which is generally a hardware or software flag. In multitasking systems, a semaphore is a variable that indicates the status of a common resource. It is used to lock the resource that is being used. A process needing the resource checks the semaphore to determine the resource's status and then decides how to proceed. A semaphore indicates to other potential users that a file or other resource is in

use and prevents access by more than one process.

[0005] However, when multiple processes need access to a common resource, problems such as starvation and/or deadlock may occur. Starvation occurs when one or more processes are prevented from accessing the common resource, and deadlock may occur when one or more processes lock each other, thereby stopping execution.

[0006] This is further complicated when there are multiple resources. The classical approach to solving this problem is to implement a semaphore for each resource. However, when the number of resources becomes very large, this solution is not very effective.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0008] FIG. 1 is a block diagram illustrating a system architecture in accordance with general embodiments of the invention.

[0009] FIG. 2 is a block diagram illustrating a prior art system for using semaphores.

[0010] FIG. 3 is a block diagram illustrating a system having self-contained semaphore system in accordance with embodiments of the invention.

[0011] FIG. 4 is a block diagram illustrating a self-contained semaphore system in one embodiment of the invention.

[0012] FIG. 5 is a flowchart illustrating a method for using the self-contained semaphore system of FIG. 4.

[0013] FIG. 6 is a block diagram illustrating a self-contained semaphore system having a timer element in another embodiment of the invention.

[0014] FIG. 7 is a flowchart illustrating a method for using the self-contained semaphore system of FIG. 6.

[0015] FIG. 8 is a block diagram illustrating a system having semaphore system that interacts with a global arbiter in accordance with embodiments of the invention.

[0016] FIG. 9 is a block diagram illustrating a global priority queue of the global arbiter of FIG. 8.

[0017] FIG. 10 is a block diagram illustrating a semaphore system that interacts with a global arbiter in one embodiment of the invention.

[0018] FIG. 11 is a sample flow diagram of how the global priority queue of the global arbiter in FIG. 8 operates.

[0019] FIG. 12 is a flowchart illustrating a method for the semaphore system of FIG. 10.

[0020] FIG. 13 is a block diagram illustrating a semaphore system having a timer element, and that interacts with a global arbiter in one embodiment of the invention.

[0021] FIG. 14 is a flowchart illustrating a method for the semaphore

system of FIG. 13.

[0022] FIG. 15 is a block diagram illustrating a sample logic circuit for the semaphore system in accordance with general embodiments of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0023] In one aspect of the invention is a method for a fairness-based semaphore system for managing access to shared system resources. The method comprises requesting a resource, and then determining if that resource is being requested by another process, or if that resource is locked by another process. If it is being requested by another process, then the resource is given to the process having a higher priority. If the resource is locked by another process, then the process is not granted access to the resource.

[0024] In variations and different embodiments, the semaphore system may comprise and use any combination of a timer element and a global arbiter having a global priority block.

[0025] The present invention includes various operations, which will be described below. The operations of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the operations. Alternatively, the operations may be performed by a combination of hardware and software.

[0026] The present invention may be provided as a computer program

product which may include a machine-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs (Compact Disc-Read Only Memories), and magneto-optical disks, ROMs (Read Only Memories), RAMs (Random Access Memories), EPROMs (Erasable Programmable Read Only Memories), EEPROMs (Electromagnetic Erasable Programmable Read Only Memories), magnetic or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions.

[0027] Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection). Accordingly, herein, a carrier wave shall be regarded as comprising a machine-readable medium.

Introduction

[0028] FIG. 1 illustrates a system 100 in which embodiments of the invention may be implemented. It comprises at least one CPU (central processing unit) 102 (only one shown), at least one memory 104 (only one shown), and a bus 106 through which the CPU 102 and memory 104 interact. The memory 104 comprises a plurality of resources 108 (only one shown), and a plurality of semaphores 110 (only one shown), and the CPU 102 comprises a plurality of processes 112 (only one shown).

[0029] FIG. 2 illustrates a prior art system 200 that that may be implemented in the system 100 of FIG. 1. In the system 200, the CPU 102 comprises a plurality of processes 202, 204, 206, and plurality of resources 208, 210, 212, 214, where each resource 208, 210, 212, 214 has a corresponding semaphore 216, 218, 220, 222 for managing shared access to the resources 208, 210, 212, 214. Alternatively, each resource 208, 210, 212, and 214 may correspond to the semaphore functionality of semaphores 216, 218, 220, 222 represented by a single semaphore object 224, rather than as separate semaphore objects 216, 218, 220, 222.

[0030] In a prior art embodiment, when a first process 202, 204, 206 wants to access a resource 208, 210, 212, 214, the first process 202, 204, 206 checks the status of the desired resource 208, 210, 212, 214 by sending a read command over the system bus 106 to the resource's 208, 210, 212, 214 corresponding semaphore 216, 218, 220, 222. If the desired resource 208, 210, 212, 214 is available, then the first process 202, 204, 206 sends a write command to the corresponding semaphore 216, 218, 220, 222 to change the status of the semaphore 216, 218, 220, 222 from unlocked to locked.

[0031] In embodiments of the invention, a system is described in which semaphores are associated with a process rather than a resource. When a semaphore is locked, it indicates the unavailability of a specified resource for the semaphore's corresponding process. When the semaphore is unlocked, it indicates that its corresponding process is not currently using any resource.

[0032] The semaphore may be changed by the occurrence of one or more events associated with its corresponding process. A semaphore system detects

the occurrence of these events and controls the status of the semaphore. In one embodiment, the semaphore systems are self-contained in that a first semaphore system arbitrates on behalf of a first process by communicating with a second semaphore that arbitrates on behalf of a second process. In another embodiment, the semaphore systems arbitrate for their respective processes, but are dependent upon a global arbiter for conflict arbitration processing. These embodiments are further described below.

Self-Contained Semaphore

[0033] In one embodiment of the invention, as illustrated in the system 300 of FIG. 3, the CPU 102 comprises a plurality of processes 302, 304, 306, and the memory 104 comprises a plurality of resources 308, 310, 312, 314, and semaphore system 316, 318, 320. In the system 300, each process 302, 304, 306 has a corresponding semaphore system 316, 318, 320 for managing shared access to the resources 308, 310, 312, 314. Alternatively, a single semaphore system 324 can be shared by all processes.

[0034] In the descriptions below, processes 302, 304, semaphore systems 316, 318, and resource 308 are depicted to illustrate the arbitration process for two processes and their corresponding semaphore systems for a given resource 308. However, one of ordinary skill in the art would understand that the descriptions apply process 306, semaphore system 320, and resource 310, 312, 314, and any other processes, semaphore systems, and resources that may be defined, but are not illustrated or discussed herein.

Semaphore Comprising Local Priority Block

[0035] FIG. 4 illustrates processes 302, 304 having corresponding semaphore systems 316, 318. Each of the semaphore systems 316, 318 comprises a semaphore 400 to indicate the availability or unavailability of a resource, and a local arbiter 402 that arbitrates for access to a given resource 308 on behalf of its corresponding process 302, 304 by monitoring events of its corresponding process 302, 304 to determine the status of the semaphore 400.

[0036] In one embodiment, a semaphore system 316, 318 may additionally comprise a local priority block 404 that has a fixed priority for its corresponding process 302, 304. Each process 302, 304 is assigned a fixed priority that gives a first process higher or lower priority than a second process. If more than one process 302, 304 requests the same resource 308 the local arbiter 402 resolves the conflict by giving the resource 308 to the process 302, 304 with the higher priority. For example, of process P1 has priority=1 and process P2 has priority=2, where 1 is a higher priority than 2, then if processes P1 and P2 simultaneously request a given resource R, the local arbiter 402 will grant process P1 resource R because P1 has a higher priority than P2.

[0037] FIG. 5 illustrates a method of the above-described embodiment. It starts at block 500 and continues to block 502 where it is determined if there is a conflict between process P1 and process P2 over a resource R (i.e., does P1's request I.D. equal P2's request I.D.?). If there is a conflict, then it is determined if P1's local priority is greater than P2's local priority at block 504. If P1's local priority is greater than P2's local priority, then access is granted to P1 at block 508. Otherwise access is denied to P1 at block 512.

[0038] If there is no conflict, then it is determined if the resource R is locked

by P2 (i.e., if P1's request I.D. does not equal P2's request I.D., then does P1's request I.D. equal P2's currently locked I.D.?) at block 506, and if P2 has released its lock on R at block 510. If R is locked by P2, and P2 has released its lock on R, then access to resource R is granted to P1 at block 508. If R is locked and P2 has not released its lock on R, then P1 is denied access to R at block 512. If there is no conflict, and the resource R is not locked by P2, then access to resource R is granted to P1 at block 508. The method ends at block 514.

Semaphore Comprising Local Arbiter Plus Timer Element

[0039] In yet another embodiment, as illustrated in FIG. 6, a semaphore system 316, 318 may comprise the local arbiter 402, plus a timer element 600. Each semaphore system 316, 318 may comprise a local priority block 404, as discussed above, as well as a timer element 600 that keeps track of the time that its corresponding process 302, 304 has waited for a given resource 308 without locking its semaphore.

[0040] In one embodiment, the timer 600 is purely time-based, where it counts the number of clock cycles that have passed since the request for a lock was given. In another embodiment, the timer 600 is a counter that determines which other processes have been granted access to the same shared resource as the one its corresponding process is requesting. In the latter embodiment, the timer 600 is incremented every time another process is granted access, and reset (for example, to 0) when its own process is granted access.

[0041] In this embodiment, the local arbiter 402 grants a first process access if it has been waiting longer than a second process. For example, if the

process P1 having priority=1 has been waiting longer than a process P2 having priority=2, then the local arbiter grants process P1 access. If process P2 has been waiting longer than process P1, then the local arbiter grants process P2 access. If both processes have been waiting an equal amount of time, then the local arbiter determines which process has the higher priority, and grants the process with the higher priority access. In this case, that is process P1.

[0042] A method illustrating this embodiment is illustrated in the flowchart of FIG. 7. The method for a process P1's semaphore system begins at block 700 and continues to block 702 where it is determined if there is a conflict between process P1 and process P2 over a resource R. If there is a conflict, then it is determined whether P1 and P2 have been waiting an equal amount of time at block 704. If there is no conflict, then it is determined if the resource is locked by process P2 at block 706.

[0043] If there is a conflict, and P1 and P2 have been waiting an equal amount of time, it is determined if P1's local priority is greater than P2's local priority at block 712. If P1's local priority is greater than P2's local priority, then access is granted to P1 at block 714. Otherwise access is denied to P1 at block 716.

[0044] If there is a conflict, and P1 and P2 have not been waiting an equal amount of time, it is determined if P1 has been waiting longer for the resource R than P2 at block 708. If P1 has been waiting longer than P2 for resource R, then access to resource R is granted to P1 at block 714. Otherwise, access to resource R is denied to P1 at block 716.

[0045] If there is no conflict, then it is determined if the resource R is locked by P2 at block 706, and if P2 has released its lock on R at block 710. If R is locked by P2, and P2 has released its lock on R, then access to resource R is granted to P1 at block 714. If R is locked and P2 has not released its lock on R, then P1 is denied access to R at block 716. If there is no conflict, and the resource R is not locked by P2, then access to resource R is granted to P1 at block 714. The method ends at block 718.

System Having Global Arbiter

[0046] In another embodiment of the invention, as illustrated in FIG. 8, the system 800 comprises a CPU having a plurality of processes 302, 304, 306, and a memory comprising a plurality of resources 308, 310, 312, 314, and a plurality of processes 302, 304, 306. In the system 300, each process 302, 304, 306 has a corresponding semaphore system 316, 318, 320 for managing shared access to the resources 308, 310, 312, 314. Alternatively, a single semaphore system 324 can be shared by all processes. Additionally, the system 800 comprises a global arbiter 802 for resolving conflicts.

[0047] In the descriptions below, processes 302, 304, semaphore systems 316, 318, and resource 308 are depicted to illustrate the arbitration process for two processes and their corresponding semaphore systems for a given resource 308. However, one of ordinary skill in the art would understand that the descriptions apply process 306, semaphore system 320, and resource 310, 312, 314, and any other processes, semaphore systems, and resources that may be defined, but are not illustrated or discussed herein.

[0048] As illustrated in FIG. 9, a global arbiter 802 may comprise a queue 900 (hereinafter "global priority queue") having entries 902, 904, 906, 908, 910, 912, 914, 916 where each entry corresponds to a priority, and each entry value indicates a unique process. (Alternatively, each entry may correspond to a unique process, and the entry value indicates the priority of the process). Initially, each process is assigned to an entry in the queue having a priority value (where the entry value is the process identifier). As each process is granted access to a resource, it is moved to the bottom of the queue such that it has the lowest priority.

[0049] A global arbiter 802 may comprise one queue for all resources such that a first process P1 that is granted access over a second process P2 subsequently has lower priority than P2 for all resources, or the global arbiter 802 may comprise a plurality of queues each corresponding to a resource such that a first process P1 that is granted access over a second process P2 subsequently has lower priority than P2 for the given resource.

[0050] For purposes of illustration, the latter of the two implementations is discussed. However, it should be understood by one of ordinary skill in the art that either implementation will work.

Semaphore Comprising Local Arbiter Only

[0051] In one embodiment, as illustrated in FIG. 10, a semaphore system 316, 318 may comprise the local arbiter 402, and each process 302, 304 is assigned an initial priority in the global priority queue 900 of the global arbiter 802. The rankings change as the processes are granted access to the resource 308.

[0052] A local arbiter 402 for a first process communicates with a local arbiter 402 for a second process to determine if there is a conflict. If there is a conflict, the global arbiter 802 resolves the conflict by giving the resource to the process with the higher global priority as determined by its global priority queue 900. For example, if process P1 has a higher priority in the global priority queue than process P2, then if processes P1 and P2 simultaneously request a given resource R, then the global arbiter grants access to the resource R to P1.

[0053] An example is illustrated in FIG. 11, where five (5) processes P1, P2, P3, P4, and P5 are at one time or another competing for a given resource, R1. Assume that the global priority queue is initialized such that P1 has the highest priority and P5 initially has the lowest priority (step 1). At step 2, P1 and P2 compete for R1 at the same time. Since P1 has higher priority than P2, P1 is granted access and then moved to the bottom of the queue in step 3.

[0054] At step 4, P4 and P5 compete for R1. Since P4 has higher priority than P5, P4 is granted access and moved to the bottom of the queue in step 5. At step 6, P1 and P4 compete for R1. Since P1 has higher priority than P4, P1 is granted access and moved to the bottom of the queue at step 7. At step 8, P2 and P1 compete for R1. Since P2 has higher priority than P1, P2 is granted access and moved to the bottom of the queue. In this manner, it is apparent that fairness is achieved since processes that have not been granted access to R1 have higher priority than processes that have been granted access.

[0055] The method discussed above is illustrated in the flowchart of FIG. 12. The method for a process P1's semaphore system begins at block 1200 and continues to block 1202 where it is determined if there is a conflict between

process P1 and process P2 over a resource R. If there is a conflict, then it is determined if P1's global priority is greater than P2's global priority at block 1204. If P1's global priority is greater than P2's global priority, then access is granted to P1 at block 1208. Otherwise access is denied to P1 at block 1212.

[0056] If there is no conflict, then it is determined if the resource R is locked by P2 at block 1206, and if P2 has released its lock on R at block 1210. If R is locked by P2, and P2 has released its lock on R, then access to resource R is granted to P1 at block 1208. If R is locked and P2 has not released its lock on R, then P1 is denied access to R at block 1212. If there is no conflict, and the resource R is not locked by P2, then access to resource R is granted to P1 at block 1208. The method ends at block 1214.

Semaphore Comprising Local Arbiter and Timer Element

[0057] In another embodiment, as illustrated in FIG. 13, a semaphore system 316, 318 may comprise the local arbiter 402 and a timer element 600 that keeps track of the time that its corresponding process has waited for a given resource without locking, *supra*. Additionally, each process is given an entry in the global priority queue 900 of the global arbiter 802.

[0058] When multiple processes request the same resource, the local arbiter 402 for a first process P1 of the multiple processes resolves a conflict between P1 and a second process P2 of the multiple processes, where P1 has higher priority than P2, as follows:

[0059] • the local arbiter 402 will grant access to the process that has been waiting longer, as determined by the timer element 600;

[0060] if both processes have been waiting the same amount of time, then arbitration is offloaded to the global arbiter 802 to resolve the conflict.

[0061] Once arbitration has been offloaded to the global arbiter 802, the global arbiter 802 consults the global priority queue 900 to determine which one of the two processes 302, 304 has higher priority. The global arbiter 802 then grants access to the process 302, 304 having the higher priority.

[0062] FIG. 14 illustrates this process. The method for a process P1's semaphore system begins at block 1400 and continues to block 1402 where it is determined if there is a conflict between process P1 and process P2 over a resource R. If there is a conflict, then it is determined whether P1 and P2 have been waiting an equal amount of time at block 1404. If there is no conflict, then it is determined if the resource is locked by process P2 at block 1406.

[0063] If there is a conflict, and P1 and P2 have been waiting an equal amount of time, it is determined if P1's global priority is greater than P2's global priority at block 1412. If P1's global priority is greater than P2's global priority, then access is granted to P1 at block 1414. Otherwise access is denied to P1 at block 1416.

[0064] If there is a conflict, and P1 and P2 have not been waiting an equal amount of time, it is determined if P1 has been waiting longer for the resource R than P2 at block 1408. If P1 has been waiting longer than P2 for resource R, then access to resource R is granted to P1 at block 1414. Otherwise, access to resource R is denied to P1 at block 1416.

[0065] If there is no conflict, then it is determined if the resource R is locked

by P2 at block 1406, and if P2 has released its lock on R at block 1410. If R is locked by P2, and P2 has released its lock on R, then access to resource R is granted to P1 at block 1414. If R is locked and P2 has not released its lock on R, then P1 is denied access to R at block 1416. If there is no conflict, and the resource R is not locked by P2, then access to resource R is granted to P1 at block 1414. The method ends at block 1418.

[0066] FIG. 15 is a basic circuit diagram for implementing a local arbiter 402 as described in embodiments of the invention. Elements 1500 and 1502 illustrate circuitry for determining whether any conflict exists. Element 1500 determines whether a first process P1 and a second process P2 are requesting the same resource R, and element 1502 determines if P1 is requesting a resource that P2 has locked. Element 1504 accepts input from the global priority block to determine which process has higher priority, and element 1506 accepts input from the timer element to determine which process has been waiting longer.

[0067] For example, for a local arbiter 402 corresponding to a process P1, access to a resource R will be granted if:

[0068] • P1 does not request the same resource as P2, and P1 does not request a resource that P2 has locked. This is represented by the circuitry of FIG. 15 as follows:

[0069] If element 1532 is TRUE, then access to R is granted to P1. If element 1532 is FALSE, then access to R is denied to P1. Element 1532 is TRUE if both elements 1508 and 1530 are TRUE.

[0070] Element 1508 generates a TRUE signal if P1 and P2 are not

requesting the same resource. Thus, if P1's request identifier does not equal P2's request identifier, then a TRUE signal is generated at element 1508.

[0071] Element 1530 generates a TRUE signal if both elements 1526 and 1528 generate a FALSE signal. Element 1528 generates a FALSE signal if either element 1512 or 1514 generates a FALSE signal. Thus, for example, element 1512 generates a FALSE signal if P1 does not request a resource that P2 has locked. Similarly, element 1526 generates a FALSE signal if either element 1510 or 1524 generates a FALSE signal. For example, element 1510 generates a FALSE signal if P1 does not request the same resource that P2 requests - in other words, this generates the opposite signal of element 1508.

[0072] • P1 does not request the same resource as P2, P1 requests a resource that P2 has locked, but P2 releases that lock. This is represented by the circuitry as follows:

[0073] If element 1532 is TRUE, then access to R is granted to P1. If element 1532 is FALSE, then access to R is denied to P1. Element 1532 is TRUE if both elements 1508 and 1530 are TRUE.

[0074] Element 1508 generates a TRUE signal if P1 and P2 are not requesting the same resource. Thus, if P1's request identifier does not equal P2's request identifier, then a TRUE signal is generated at element 1508.

[0075] Element 1528 generates a FALSE signal if either element 1512 or 1514 generates a FALSE signal. Thus, for example, if element 1512 generates a TRUE signal instead (because P1 requests a resource that P2 has locked), then element 1514 must generate a FALSE signal. This occurs if P2 releases its

currently locked identifier. Similarly, element 1526 generates a FALSE signal if either element 1510 or 1524 generates a FALSE signal. For example, element 1510 generates a FALSE signal if P1 does not request the same resource that P2 requests - in other words, this generates the opposite signal of element 1508.

[0076] • P1 requests the same resource as P2, P1 is not requesting a resource that P2 as locked, (or if P1 has requested a resource that P2 has locked, then P2 has released that lock), and P1 has waited longer for the resource than P2. This is represented by the circuitry of FIG. 15 as follows:

[0077] If element 1530 is TRUE, then access to R is granted to P1. If element 1530 is FALSE, then access to R is denied to P1. Element 1530 is TRUE if both elements 1526 and 1528 are FALSE.

[0078] Element 1528 generates a FALSE signal if either element 1512 or 1514 generates a FALSE signal. Thus, for example, element 1512 generates a FALSE signal if P1 does not request a resource that P2 has locked. Also, if element 1512 generates a TRUE signal instead (because P1 requests a resource that P2 has locked), then element 1514 must generate a FALSE signal. This occurs if P2 releases its currently locked identifier.

[0079] Since element 1510 generates a TRUE signal if P1 has requested the same resource as P2, element 1526 generates a FALSE signal under the following circumstances. P1 and P2 have not been waiting the same amount of time, causing element 1518 to generate a FALSE signal, and therefore, causing element 1520 to generate a FALSE signal. Since P2 has not been waiting longer than P1, element 1522 generates a FALSE signal. Consequently, element 1524

generates a FALSE signal, and element 1526 generates a FALSE signal. As a result, element 1530 generates a TRUE signal, and P1 is granted access to R.

[0080] • P1 requests the same resource as P2, P1 is not requesting a resource that P2 as locked, (or if P1 has requested a resource that P2 has locked, then P2 has released that lock), P1 has waited for the resource equally as long as P2, and P1's global priority is greater than P2's global priority. This is represented by the circuitry of FIG. 15 as follows:

[0081] If element 1530 is TRUE, then access to R is granted to P1. If element 1530 is FALSE, then access to R is denied to P1. Element 1530 is TRUE if both elements 1526 and 1528 are FALSE.

[0082] Element 1528 generates a FALSE signal if either element 1512 or 1514 generates a FALSE signal. Thus, for example, element 1512 generates a FALSE signal if P1 does not request a resource that P2 has locked. Also, if element 1512 generates a TRUE signal instead (because P1 requests a resource that P2 has locked), then element 1514 must generate a FALSE signal. This occurs if P2 releases its currently locked identifier.

[0083] Since element 1510 generates a TRUE signal if P1 has requested the same resource as P2, element 1526 generates a FALSE signal only if element 1524 generates a FALSE signal. P1 and P2 have been waiting the same amount of time, causing element 1518 to generate a TRUE signal. Since element 1518 is TRUE, element 1522 is necessarily FALSE. P2 does not have a higher priority than P1 so element 1516 is FALSE causing element 1520 to be FALSE. This causes element 1524 to be FALSE, and, therefore, element 1526. As a result,

element 1530 generates a TRUE signal, and P1 is granted access to R.

Conclusion

[0084] Thus, an invention has been described for a semaphore system that is based on process events rather than on resource events. In one embodiment, the process event is a simple fixed priority. In another embodiment, the process event is a fairness-based dynamic priority.

[0085] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

WHAT IS CLAIMED IS:

1. A method comprising:

requesting access to a shared resource for a first process having a first local priority;

determining if a second process is simultaneously requesting access to the shared resource, the second process having a second local priority; and

if the second process is simultaneously requesting access to the shared resource, then granting access one of the first priority and the second priority having a higher local priority.

2. The method of claim 1, wherein the local priority is fixed for each of the first and the second process.

3. The method of claim 1, additionally comprising if the second process is not simultaneously requesting access to the shared resource, then:

determining if the second process currently has a lock on the shared resource;

if the second process currently has a lock on the shared resource, then denying the first process access to the shared resource; and

if the second process does not have a lock on the shared resource, then granting the first process access to the shared resource.

4. A method comprising:

requesting access to a shared resource for a first process having a first local priority, and a first wait time;

determining if a second process is simultaneously requesting access to the shared resource, the second process having a second local priority, and a second wait time;

if the second process is simultaneously requesting access to the shared resource, then granting access to one of the first process and the second process having a longer wait time; and

if the first wait time equals the second wait time, then granting access to one of the first process and the second process having a local higher priority.

5. The method of claim 4, wherein the local priority is fixed for each of the first and the second process.

6. The method of claim 4, additionally comprising if the second process is not simultaneously requesting access to the shared resource, then:

determining if the second process currently has a lock on the shared resource;

if the second process currently has a lock on the shared resource, then denying the first process access to the shared resource; and

if the second process does not have a lock on the shared resource, then granting the first process access to the shared resource.

7. A method comprising:

requesting access to a shared resource for a first process having a first global priority on a global priority queue of a global arbiter;

determining if a second process is simultaneously requesting access to the shared resource, the second process having a second global priority on the global priority queue of the global arbiter; and

if the second process is simultaneously requesting access to the shared resource, then granting access to one of the first process and the second process having a higher global priority.

8. The method of claim 7, wherein the global priority queue is one of a plurality of global priority queues in the global arbiter, and each global priority queue corresponds to a given shared resource.

9. The method of claim 7, additionally comprising if the second process is not simultaneously requesting access to the shared resource, then:

determining if the second process currently has a lock on the shared resource;

if the second process currently has a lock on the shared resource, then denying the first process access to the shared resource; and

if the second process does not have a lock on the shared resource, then granting the first process access to the shared resource.

10. A method comprising:

requesting access to a shared resource for a first process having a first global priority on a global priority queue of a global arbiter, and having a first wait time;

determining if a second process is simultaneously requesting access to the shared resource, the second process having a second global priority on the global priority queue of the global arbiter, and having a second wait time;

if the second process is simultaneously requesting access to the shared resource, then granting access to one of the first process and the second process having a longer wait time; and

if the first wait time is equal to the second wait time, then granting access to one of the first process and the second process having a higher than global priority.

11. The method of claim 10, wherein the global priority queue is one of a plurality of global priority queues in the global arbiter, and each global priority queue corresponds to a given shared resource.

12. The method of claim 10, additionally comprising if the second process is not simultaneously requesting access to the shared resource, then:

determining if the second process currently has a lock on the shared resource;

if the second process currently has a lock on the shared resource, then denying the first process access to the shared resource; and

if the second process does not have a lock on the shared resource, then granting the first process access to the shared resource.

13. An apparatus comprising:

a local arbiter to arbitrate on behalf of the corresponding process for one of a plurality of resources; and

a semaphore to indicate a status of the corresponding process.

14. The apparatus of claim 13, additionally comprising a local priority block to indicate a local priority of the corresponding process.

15. The apparatus of claim 13, additionally comprising a timer element to determine a wait time for the corresponding process.

16. A system comprising:

one or more shared resources; and

one or more processes, each corresponding to a semaphore system, and each semaphore system having a local arbiter to arbitrate for access to a given one of the shared resources.

17. The system as in claim 16, wherein a given semaphore system additionally comprises a local arbiter block having a local priority corresponding to a corresponding process, and the local arbiter arbitrates for access to a given one of the shared resources by granting access to the corresponding process if its corresponding process has a local global priority than a conflicting process.

18. The system of claim 17, wherein the local priority is fixed.

19. The system as in claim 17, wherein the given semaphore system additionally comprises a timer element, and the local arbiter arbitrates for access to a given one of the shared resource by:

granting access to the corresponding process if the corresponding process waited longer for the given resource than the conflicting process; and

if the corresponding process waited the same amount of time for the given resource as the conflicting process, then granting access to the corresponding process if the corresponding process has a higher local priority than the conflicting process.

20. The system of claim 16, wherein the local arbiter arbitrates for access to a given one of the shared resources by granting access to the corresponding process if there are no conflicting processes.

21. The system as in claim 20, the system additionally comprising a global arbiter having a global priority queue, the global arbiter to:

modify process priorities by moving processes that have been granted access to a given resource to a position in the global priority queue having a lowest priority; and

arbitrate conflicts between a first process and a second process by granting access to one of the first process and the second process having a having a higher global priority.

22. The system of claim 16, wherein the semaphore additionally comprises a timer element, and the local arbiter arbitrates for access to a given one of the shared resources by:

granting access to the corresponding process if the corresponding process waited longer for the given resource than a conflicting process; and

if the corresponding process has waited the same amount of time for the given resource as the conflicting process, then offloading the arbitration process to a global arbiter.

23. The system as in claim 22, the system additionally comprising the global arbiter having a global priority queue, the global arbiter to:

modify priorities to processes by moving processes that have been granted access to a given resource to a position in the global priority queue having a lowest priority; and

arbitrate conflicts between a first process and a second process by granting

access to one of the first process and the second process having a higher priority.

24. A machine-readable medium having stored thereon data representing sequences of instructions, the sequences of instructions which, when executed by a processor, cause the processor to perform the following:

request access to a shared resource for a first process having a first local priority;

determine if a second process is simultaneously requesting access to the shared resource, the second process having a second local priority; and

if the second process simultaneously requests access to the shared resource, then grant access one of the first priority and the second priority having a higher local priority.

25. The machine-readable medium of claim 24, wherein the local priority is fixed for each of the first and the second process.

26. The machine-readable medium of claim 24, additionally comprising if the second process is not simultaneously requesting access to the shared resource, then additionally comprising sequences of instructions which, when executed by a processor, cause the processor to perform:

determine if the second process currently has a lock on the shared resource;

if the second process currently has a lock on the shared resource, then

deny the first process access to the shared resource; and

if the second process does not have a lock on the shared resource, then grant the first process access to the shared resource.

27. An apparatus comprising:

at least one processor; and

a machine-readable medium having instructions encoded thereon, which when executed by the processor, are capable of directing the processor to:

request access to a shared resource for a first process having a first local priority;

determine if a second process is simultaneously requesting access to the shared resource, the second process having a second local priority; and

if the second process simultaneously requests access to the shared resource, then grant access one of the first priority and the second priority having a higher local priority.

28. The apparatus of claim 27, wherein the local priority is fixed for each of the first and the second process.

29. The apparatus of claim 27, additionally comprising if the second process is not simultaneously requesting access to the shared resource, then additionally encoded instructions which, when executed by a processor, are capable of causing the processor to:

determine if the second process currently has a lock on the shared resource;

if the second process currently has a lock on the shared resource, then deny the first process access to the shared resource; and

if the second process does not have a lock on the shared resource, then grant the first process access to the shared resource.

30. An apparatus comprising:

means for arbitrating on behalf of the corresponding process for one of a plurality of resources; and

means for indicating a status of the corresponding process.

31. The apparatus of claim 30, additionally comprising means for indicating a local priority of the corresponding process.

32. The apparatus of claim 30, additionally comprising means for determining a wait time for the corresponding process.