

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
8 March 2001 (08.03.2001)

PCT

(10) International Publication Number
WO 01/16740 A2

(51) International Patent Classification⁷: **G06F 9/52**

[US/US]; 2801 Wells Branch Parkway #2128, Austin, TX 78728 (US).

(21) International Application Number: PCT/US00/24210

(22) International Filing Date: 31 August 2000 (31.08.2000)

(74) Agent: **BRUCKNER, John, J.**; Fulbright & Jaworski, L.L.P., Suite 2400, 600 Congress Avenue, Austin, TX 78701 (US).

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/152,151	31 August 1999 (31.08.1999)	US
60/220,794	26 July 2000 (26.07.2000)	US
60/220,748	26 July 2000 (26.07.2000)	US

(81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier applications:

US	60/152,151 (CIP)
Filed on	31 August 1999 (31.08.1999)
US	60/220,794 (CIP)
Filed on	26 July 2000 (26.07.2000)
US	60/220,748 (CIP)
Filed on	26 July 2000 (26.07.2000)

(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— Without international search report and to be republished upon receipt of that report.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(71) Applicant (*for all designated States except US*): **TIMES N SYSTEMS, INC.** [US/US]; Building B, Suite P, 1908 Kramer Lane, Austin, TX 78758 (US).

(72) Inventor; and

(75) Inventor/Applicant (*for US only*): **WEST, Karlon, K.**

(54) Title: EFFICIENT EVENT WAITING

(57) Abstract: Methods, systems and devices are described for efficient event waiting. A method includes: waiting for an event lock release including: providing mutually exclusive access primitives to ensure data or resource integrity; recording an intention to wait; and then performing other processing. The methods, systems and devices provide advantages because the speed and scalability of parallel processor systems is enhanced.



WO 01/16740 A2

EFFICIENT EVENT WAITING

BACKGROUND OF THE INVENTION

5 1. Field of the Invention

 The invention relates generally to the field of computer systems, and more specifically to computer systems where two or more CPU are connected to one or more RAM subsystems, or portions thereof, where each CPU can access a portion of the computing system where mutually exclusive access primitives
10 are needed among a plurality of CPUs. More particularly, the invention relates to computer science techniques that utilize efficient event waiting.

2. Discussion of the Related Art

 The clustering of workstations is a well-known art. In the most common cases, the clustering involves workstations that operate almost totally
15 independently, utilizing the network only to share such services as a printer, license-limited applications, or shared files.

 In more-closely-coupled environments, some software packages (such as NQS) allow a cluster of workstations to share work. In such cases the work arrives, typically as batch jobs, at an entry point to the cluster where it is queued
20 and dispatched to the workstations on the basis of load.

 In both of these cases, and all other known cases of clustering, the operating system and cluster subsystem are built around the concept of message-passing. The term message-passing means that a given workstation operates on some portion of a job until communications (to send or receive data,
25 typically) with another workstation is necessary. Then, the first workstation prepares and communicates with the other workstation.

 Another well-known art is that of clustering processors within a machine, usually called a Massively Parallel Processor or MPP, in which the techniques are essentially identical to those of clustered workstations. Usually,
30 the bandwidth and latency of the interconnect network of an MPP are more highly optimized, but the system operation is the same.

In the general case, the passing of a message is an extremely expensive operation; expensive in the sense that many CPU cycles in the sender and receiver are consumed by the process of sending, receiving, bracketing, verifying, and routing the message, CPU cycles that are therefore not available
5 for other operations. A highly streamlined message-passing subsystem can typically require 10,000 to 20,000 CPU cycles or more.

There are specific cases wherein the passing of a message requires significantly less overhead. However, none of these specific cases is adaptable to a general-purpose computer system.

10 Message-passing parallel processor systems have been offered commercially for years but have failed to capture significant market share because of poor performance and difficulty of programming for typical parallel applications. Message-passing parallel processor systems do have some advantages. In particular, because they share no resources, message-passing
15 parallel processor systems are easier to provide with high-availability features. What is needed is a better approach to parallel processor systems.

There are alternatives to the passing of messages for closely-coupled cluster work. One such alternative is the use of shared memory for inter-processor communication.

20 Shared-memory systems, have been much more successful at capturing market share than message-passing systems because of the dramatically superior performance of shared-memory systems, up to about four-processor systems. In Search of Clusters, Gregory F. Pfister 2nd ed. (January 1998) Prentice Hall Computer Books, ISBN: 0138997098 describes a computing
25 system with multiple processing nodes in which each processing node is provided with private, local memory and also has access to a range of memory which is shared with other processing nodes. The disclosure of this publication in its entirety is hereby expressly incorporated herein by reference for the purpose of indicating the background of the invention and illustrating the state
30 of the art.

However, providing high availability for traditional shared-memory systems has proved to be an elusive goal. The nature of these systems, which

share all code and all data, including that data which controls the shared operating systems, is incompatible with the separation normally required for high availability. What is needed is an approach to shared-memory systems that improves availability.

5 Although the use of shared memory for inter-processor communication is a well-known art, prior to the teachings of U.S. Ser. No. 09/273,430, filed March 19, 1999, entitled Shared Memory Apparatus and Method for Multiprocessing Systems, the processors shared a single copy of the operating system. The problem with such systems is that they cannot be efficiently scaled
10 beyond four to eight way systems except in unusual circumstances. All known cases of said unusual circumstances are such that the systems are not good price-performance systems for general-purpose computing.

 The entire contents of U.S. Patent Applications 09/273,430, filed March 19, 1999 and PCT/US00/01262, filed January 18, 2000 are hereby expressly
15 incorporated by reference herein for all purposes. U.S. Ser. No. 09/273,430, improved upon the concept of shared memory by teaching the concept which will herein be referred to as a tight cluster. The concept of a tight cluster is that of individual computers, each with its own CPU(s), memory, I/O, and operating system, but for which collection of computers there is a portion of memory
20 which is shared by all the computers and via which they can exchange information. U.S. Ser. No. 09/273,430 describes a system in which each processing node is provided with its own private copy of an operating system and in which the connection to shared memory is via a standard bus. The advantage of a tight cluster in comparison to an SMP is "scalability" which
25 means that a much larger number of computers can be attached together via a tight cluster than an SMP with little loss of processing efficiency.

 What is needed are improvements to the concept of the tight cluster. What is also needed is an expansion of the concept of the tight cluster.

 In a typical computing system, with two or more CPU, tasks can be
30 performed simultaneously on each CPU. Software can take advantage of this property, and have multiple tasks working at the same time, possibly accessing the same shared resources, such as data structures in memory, or physical

devices connected to the computing system. To ensure resource integrity, certain portions of a given task may need to complete without the possibility of another CPU performing certain tasks at the same time. Synchronization primitives are usually provided by the operating system to handle these cases.

5 When one CPU enters a synchronization primitive (such as a lock or event) first, it is given access. When another CPU then tries to enter the same synchronization primitive, it will usually try a few times, and then cause the currently running task to be suspended so that it can run some other task while the first CPU still owns that lock or event. There is a lot of system overhead in
10 preparing a task to be suspended, saving the task's context, determining the precise location to be suspended, updating system data structures, invoking the task scheduler, etc. Also, certain types of synchronization primitives are designed to be held for extremely short periods of time.

 Thus, a problem in any shared-memory, multiprocessor system is the
15 posting and waiting for events. The prior art teaches that if a first process is utilizing a first semaphore, and therefore has it locked, a second process may require said semaphore for processing required to follow.

 The prior art further teaches that said second process can wait for the lock to become free by a technique called a "spin-wait". A spin-wait is defined
20 as having said second process execute those instructions necessary to secure the lock, check to see if the attempt was successful, and then if not immediately repeat the loop, thus "spinning" through a routine until the lock becomes free, after which said second process, having secured the lock, continues with productive processing. This assures that said second process will obtain the lock
25 with very small delay after it is released, but it also assures that the second processor, on which said second process is running, will do no productive work for the entire duration of the spin, which can be many thousands of cycles.

 An alternative taught in the prior art is that said second process can
"sleep-wait" for the lock to become free. In such implementations, the freeing of
30 the lock by said first processes will post an "event" to the kernel when said lock is freed, and the kernel will then check a queue of sleeping processes waiting for locks to determine if any is waiting on said lock. If so, the kernel dispatches

said process so that it can acquire the lock and then proceed with processing. This alternative technique is superior in some ways to the spin-wait technique, but is inferior in other ways. First, there are many "barrier" conditions in multiprocessing systems such that the time a second process seeks to acquire a lock is approximately the same time a first-process is releasing it. Sleeping consumes no system resources, but going to sleep, and being awakened both consume significant processing time (thousands of cycles). Therefore, a second process may go to sleep only a few cycles before a lock is freed, thus losing significant elapsed time before proceeding, and also consuming thousands of cycles in kernel time managing the queuing and dispatching thereof.

The second technique also involves another drawback. If a lock is taken, said second process finds it locked, then does necessary housekeeping (releasing other locks and resources, saving state information) then enters the event-wait queue. It is possible (and does occur) that the lock becomes free while said second process is still in the mode of preparing to wait but before having been recorded as an entry in the event-lock queue. The system can then receive the event signal from said first process, check the event-waiting queue for processes waiting for that lock, find none there, and continue with other processing. Said second process, upon then entering the queue, will not be signaled of the event and will then wait a very long time till secondary mechanisms restore it to operational status.

Such delays can significantly reduce performance for processes which are exchanging information on a frequent basis. What is needed, therefore, is an approach to waiting that is more efficient.

SUMMARY OF THE INVENTION

A goal of the invention is to simultaneously satisfy the above-discussed requirements of improving and expanding the tight cluster concept which, in the case of the prior art, are not satisfied.

One embodiment of the invention is based on a method, comprising: waiting for an event lock release including: providing mutually exclusive access primitives to ensure data or resource integrity; recording an intention to wait;

and then performing other processing. Another embodiment of the invention is based on an apparatus, comprising: a shared memory unit; a first processing node coupled to said shared memory unit; and a second processing node coupled to said shared memory unit, wherein mutually exclusive access primitives ensure data or resource integrity and, if said first processing node holds an event lock that said second processing node has requested, said second processing node records an intention to wait entry in an event queue before performing housekeeping necessary to actually wait and marking the intention to wait entry to indicate that waiting has not actually commenced. Another embodiment of the invention is based on an electronic media, comprising: a computer program adapted to wait for an event lock release by: providing mutually exclusive access primitives to ensure data or resource integrity; recording an intention to wait; and then performing other processing. Another embodiment of the invention is based on a computer program comprising computer program means adapted to perform the steps of waiting for an event lock release including: providing mutually exclusive access primitives to ensure data or resource integrity; recording an intention to wait; and then performing other processing when said computer program is run on a computer. Another embodiment of the invention is based on a system, comprising a multiplicity of processors, sharing at least some memory, capable of executing a multiplicity of processes in parallel, and including semaphore means by which one of said processes may lock said semaphore or other resource to preclude other processes from modifying certain data while said semaphore is locked, and further capable of issuing a signal called to one or more operating system routines when said locked semaphore is unlocked. Another embodiment of the invention is based on a computer system in which each of two or more CPUs has access to shared synchronization primitives. Another embodiment of the invention is based on a computer system, comprising: shared memory unit; first processing node coupled to said shared memory unit, said first processing node including a first private memory and a first operating system; and a second processing node coupled to said shared memory unit, said second processing node including a second private memory and a second operating system,

wherein one or more synchronization primitives provide suspension of a task. Another embodiment of the invention is based on a computer system, comprising: shared memory unit; first processing node coupled to said shared memory unit, said first processing node including a first private memory and a first operating system; and second processing node coupled to said shared memory unit, said second processing node including a second private memory and a second operating system, wherein a synchronization primitive is tried one more time before a final task suspension is implemented.

These, and other goals and embodiments of the invention will be better appreciated and understood when considered in conjunction with the following description and the accompanying drawing. It should be understood, however, that the following description, while indicating preferred embodiments of the invention and numerous specific details thereof, is given by way of illustration and not of limitation. Many changes and modifications may be made within the scope of the invention without departing from the spirit thereof, and the invention includes all such modifications.

BRIEF DESCRIPTION OF THE DRAWING

A clear conception of the advantages and features constituting the invention, and of the components and operation of model systems provided with the invention, will become more readily apparent by referring to the exemplary, and therefore nonlimiting, embodiments illustrated in the drawing accompanying and forming a part of this specification, wherein like reference characters (if they occur in more than one view) designate the same parts. It should be noted that the features illustrated in the drawing are not necessarily drawn to scale.

FIG. 1 illustrates a block schematic view of a system, representing an embodiment of the invention.

DESCRIPTION OF PREFERRED EMBODIMENTS

The invention and the various features and advantageous details thereof are explained more fully with reference to the nonlimiting embodiments that are

illustrated in the accompanying drawing and detailed in the following description of preferred embodiments. Descriptions of well known components and processing techniques are omitted so as not to unnecessarily obscure the invention in detail.

5 The teachings of U.S. Ser. No. 09/273,430 include a system which is a single entity; one large supercomputer. The invention is also applicable to a cluster of workstations, or even a network.

 The invention is applicable to systems of the type of Pfister or the type of U.S. Ser. No. 09/273,430 in which each processing node has its own copy of an operating system. The invention is also applicable to other types of multiple
10 processing node systems.

 The context of the invention can include a tight cluster as described in U.S. Ser. No. 09/273,430. A tight cluster is defined as a cluster of workstations or an arrangement within a single, multiple-processor machine in which the
15 processors are connected by a high-speed, low-latency interconnection, and in which some but not all memory is shared among the processors. Within the scope of a given processor, accesses to a first set of ranges of memory addresses will be to local, private memory but accesses to a second set of memory address ranges will be to shared memory. The significant advantage to a tight cluster in
20 comparison to a message-passing cluster is that, assuming the environment has been appropriately established, the exchange of information involves a single STORE instruction by the sending processor and a subsequent single LOAD instruction by the receiving processor.

 The establishment of the environment, taught by U.S. Ser. No.
25 09/273,430 and more fully by companion disclosures (U.S. Provisional Application Ser. No. 60/220,794, filed July 26, 2000; U.S. Provisional Application Ser. No. 60/220,748, filed July 26, 2000; WSGR 15245-711; WSGR 15245-712; WSGR 15245-715; WSGR 15245-716; WSGR 15245-717; WSGR 15245-718; WSGR 15245-719; and WSGR 15245-720, the entire
30 contents of all which are hereby expressly incorporated herein by reference for all purposes) can be performed in such a way as to require relatively little system overhead, and to be done once for many, many information exchanges.

Therefore, a comparison of 10,000 instructions for message-passing to a pair of instructions for tight-clustering, is valid.

Among the approaches to controlling shared memory in a tight cluster for improved performance is the provision of a highly efficient method of waiting for event locks. The prior art teaches methods involving early commitment to the lock, teaches methods of spin-waiting for the lock, and also teaches methods of sleep-waiting for the lock. This invention teaches an entirely new concept - the concept of recording "intention to wait", followed by other processing, and then followed if necessary by traditional techniques.

A problem solved is that all of the prior methods are extremely inefficient in that at least one of the subsystems involved expends much resource in waiting, sleeping, or polling. The invention taught here is such that such expense is often avoided.

In the context of a computing system for which there exists two or more central processing units (CPU) and for which there is some common computing resource, such as the Memory Subsystem (RAM) or portion thereof, the invention can include providing mutually exclusive access primitives to ensure data or resource integrity. If a CPU is going to wait on a lock or event, the CPU can declare the intention to wait, perform pre-wait tasks, then make a final attempt to acquire the resource before suspending.

The invention can be used in an environment as described in U.S. Ser. No. 09/273, 430 where multiple computers are provided with means to selectively address a first set of memory address ranges which will be to private memory and a second set of memory ranges which will be to shared memory.

In such an environment, spin-waits and sleep-waits can cause significant processing delays. This invention therefore teaches a new concept, the "intention to wait". The concept is that when a first process has a lock and a second process encounters the need for the lock, said second process records, by any of several means, an "intent to wait". In a preferred embodiment, the recording is done by placing an entry in the event queue immediately, before performing the housekeeping necessary to actually sleep-wait, and marking the entry to indicate that waiting has not actually commenced.

Said second process may then spin-wait for a few cycles or may immediately begin the housekeeping tasks. When the housekeeping tasks are complete, said second process then may check one last time for the lock then sleep, or may directly enter the dormant (wait) state. There are a number of
5 methods of recording and managing "intent to wait" that are within the scope and teaching of this invention.

Referring to FIG. 1 an embodiment of the invention begins with a processing node attempting to acquire a lock at block 101. At block 102 the invention determines if the lock is held. If the lock is not held, the processing
10 node acquires the lock at block 103. If the lock is held, the processing node determines if ten attempts have been made to acquire the lock at block 104. If ten attempts have not been made, the node returns to block 101. If ten attempts have been made, the node begins a task suspension code at block 105. Block 105 includes recording an intention to wait. After block 105 is complete, the
15 node makes one more determination of whether the lock is held at block 106. If the lock is not held, the processing node goes to block 103 and acquires the lock. If the lock is held, the processing node enters a suspend task block 107.

When the event (lock release) is actually signaled, the kernel checks the event queue for processes awaiting the event, and also checks for processes
20 which are intending to await the event. Should there be members of the latter, the kernel records for said intending-to-wait process that the event has occurred, and said process checks said record just before going to sleep. If the event has already occurred at said checking time, said process abandons the final path to dormancy but instead acquires the lock and resumes productive processing.

25 In a compute system where two or more CPUs have access to common computing resources, tasks running on each CPU sometimes require synchronization primitives (locks, events, etc) to ensure resource integrity. Certain primitives, such as spinlocks and fast events are designed to be very efficient, and to be held by any one task for a very small period of time.

30 However, a given lock or event may be held for a long time, so the ability for the waiting CPU to suspend a task waiting on the lock or event and to run another task increases overall system performance. However, when the

locks are held for brief periods of time, the system overhead of suspending and switching tasks repeatedly adversely affect system performance.

When a task tries to acquire a lock or event, if the lock or event is not held, the task is granted immediate access and continues execution. When
5 another CPU runs a task that requires the same lock or event, there are several standard implementations:

First, the second CPU can spin forever on the lock or event, which will provide the second task the fastest time to acquire the lock or event, but will dramatically lower overall system performance since the second CPU is not
10 performing any useful work.

Second, the second CPU can try once, then if unsuccessful, immediately suspend the second task and go do other work. This can increase overall system performance but may greatly reduce the performance of the individual application.

15 Third, the second CPU can try for awhile, then if still unsuccessful, suspend the second task and go do other work.

This invention addresses the last two methods. In order to suspend a task, the CPU must save the context of the task, including program counter, stack registers, etc., determine the priority of the task in order to place the task
20 on the correct suspend queue, search the list of tasks ready to run to find the highest priority ready to run task, and finally enqueue the suspending task, dequeue the ready task, and begin running. All of that work to prepare to suspend a task takes time, time which the other CPU running the other task may have released the lock or event. Therefore, preferred embodiments declare an
25 intention to acquire a lock or event, or other synchronization primitive, and for attempting to acquire the lock a single time before the final act of suspending the current task and switching to a new task.

U.S. Ser. No. 09/273,430 described a system in which each compute node has its own, private memory, but in which there is also provided a shared
30 global memory and a shared global lock address space, accessible by all compute nodes. In this case, contention for shared locks only occurs when more than one node is attempting to access some shared lock at the same time. It is

also possible in a NUMA-based compute system, where some form of global synchronization primitives are provided, as well as in a standard SMP system, that the means taught by this disclosure are applicable.

While not being limited to any particular performance indicator or diagnostic identifier, preferred embodiments of the invention can be identified one at a time by testing for the substantially highest performance. The test for the substantially highest performance can be carried out without undue experimentation by the use of a simple and conventional benchmark (speed) experiment.

The term substantially, as used herein, is defined as at least approaching a given state (e.g., preferably within 10% of, more preferably within 1% of, and most preferably within 0.1% of). The term coupled, as used herein, is defined as connected, although not necessarily directly, and not necessarily mechanically. The term means, as used herein, is defined as hardware, firmware and/or software for achieving a result. The term program or phrase computer program, as used herein, is defined as a sequence of instructions designed for execution on a computer system. A program may include a subroutine, a function, a procedure, an object method, an object implementation, an executable application, an applet, a servlet, a source code, an object code, and/or other sequence of instructions designed for execution on a computer system.

Practical Applications of the Invention

A practical application of the invention that has value within the technological arts is an environment where are multiple compute nodes, each with one or more CPU and each CPU with local RAM, and where there are one or more RAM units which are accessible by some or all of the compute nodes. Another practical application of the invention that has value within the technological arts is waveform transformation. Further, the invention is useful in conjunction with data input and transformation (such as are used for the purpose of speech recognition), or in conjunction with transforming the appearance of a display (such as are used for the purpose of video games), or the like. There are virtually innumerable uses for the invention, all of which need not be detailed here.

Advantages of the Invention

A system, representing an embodiment of the invention, can be cost effective and advantageous for at least the following reasons. The invention improves the speed of parallel computing systems. The invention improves the scalability of parallel computing systems.

All the disclosed embodiments of the invention described herein can be realized and practiced without undue experimentation. Although the best mode of carrying out the invention contemplated by the inventors is disclosed above, practice of the invention is not limited thereto. Accordingly, it will be appreciated by those skilled in the art that the invention may be practiced otherwise than as specifically described herein.

For example, although the efficient event waiting described herein can be a separate module, it will be manifest that the efficient event waiting may be integrated into the system with which it is associated. Furthermore, all the disclosed elements and features of each disclosed embodiment can be combined with, or substituted for, the disclosed elements and features of every other disclosed embodiment except where such elements or features are mutually exclusive.

It will be manifest that various additions, modifications and rearrangements of the features of the invention may be made without deviating from the spirit and scope of the underlying inventive concept. It is intended that the scope of the invention as defined by the appended claims and their equivalents cover all such additions, modifications, and rearrangements.

The appended claims are not to be interpreted as including means-plus-function limitations, unless such a limitation is explicitly recited in a given claim using the phrase “means for.” Expedient embodiments of the invention are differentiated by the appended subclaims.

CLAIMS

What is claimed is:

1. A method, comprising:
5 waiting for an event lock release including:
 providing mutually exclusive access primitives to ensure data or
 resource integrity;
 recording an intention to wait; and then
 performing other processing.
10
2. The method of claim 1, wherein recording said intention to wait includes
placing an intention to wait entry in an event queue before performing
housekeeping necessary to actually wait and marking the intention to wait entry
to indicate that waiting has not actually commenced.
15
3. The method of claim 1, further comprising waiting after performing
other processing.
4. The method of claim 3, wherein waiting includes sleep-waiting.
20
5. The method of claim 3, wherein waiting includes spin-waiting.
6. The method of claim 1, wherein performing other processing includes
making a final attempt to acquire the lock release after performing other
25 processing and before waiting.
7. The method of claim 1, further comprising repeatedly attempting to
acquire the lock release before recording the intention to wait.
- 30 8. An apparatus, comprising:
 a shared memory unit;
 a first processing node coupled to said shared memory unit; and

a second processing node coupled to said shared memory unit,
wherein mutually exclusive access primitives ensure data or resource
integrity and, if said first processing node holds an event lock that said second
processing node has requested, said second processing node records an intention
5 to wait entry in an event queue before performing housekeeping necessary to
actually wait and marking the intention to wait entry to indicate that waiting has
not actually commenced.

9. A computer system comprising the apparatus of claim 8.

10. An electronic media, comprising: a computer program adapted to wait
for an event lock release by: providing mutually exclusive access primitives to
ensure data or resource integrity; recording an intention to wait; and then
performing other processing.

11. A computer program comprising computer program means adapted to
perform the steps of waiting for an event lock release including: providing
mutually exclusive access primitives to ensure data or resource integrity;
recording an intention to wait; and then performing other processing when said
20 computer program is run on a computer.

12. A computer program as claimed in claim 11, embodied on a computer-
readable medium.

13. A system, comprising a multiplicity of processors, sharing at least some
memory, capable of executing a multiplicity of processes in parallel, and
including semaphore means by which one of said processes may lock said
semaphore or other resource to preclude other processes from modifying certain
data while said semaphore is locked, and further capable of issuing a signal
30 called to one or more operating system routines when said locked semaphore is
unlocked.

14. The system of claim 13, further comprising operating systems which respond to said signals by reading control information indicating whether there are dormant processes which are waiting on said semaphore or other resource and by returning to active status such of said processes that are awaiting the particular event of the unlocking of said locked semaphore.

15. The system of claim 14, further comprising means for active processes to check whether semaphores are unlocked, and with means for said active processes to repeatedly check or to enter a dormant state, within which said now-dormant processes are provided with means responsive to said events, said means sufficient for said operating system to restore said dormant process to an active state.

16. The system of claim 15, further comprising means by which an active process may record "intent to wait" while still active, and means by which said operating system may notify still-active processes that the event awaited has occurred, and means by which said still-active processes may secure said lock and continue processing rather than continuing the path to a dormant state.

17. A computer system in which each of two or more CPUs has access to shared synchronization primitives.

18. A computer system as described in claim 17 where one or more synchronization primitives provide for the suspension of a task.

19. A computer system as described in claim 17 where trying a synchronization primitive one more time before final task suspension is implemented.

20. A computer system, comprising:
a shared memory unit;

a first processing node coupled to said shared memory unit, said first processing node including a first private memory and a first operating system;
and

5 a second processing node coupled to said shared memory unit, said second processing node including a second private memory and a second operating system,
wherein one or more synchronization primitives provide suspension of a task.

21. A computer system, comprising:
- 10 a shared memory unit;
a first processing node coupled to said shared memory unit, said first processing node including a first private memory and a first operating system;
and
a second processing node coupled to said shared memory unit, said
15 second processing node including a second private memory and a second operating system,
wherein a synchronization primitive is tried one more time before a final task suspension is implemented.

1/1

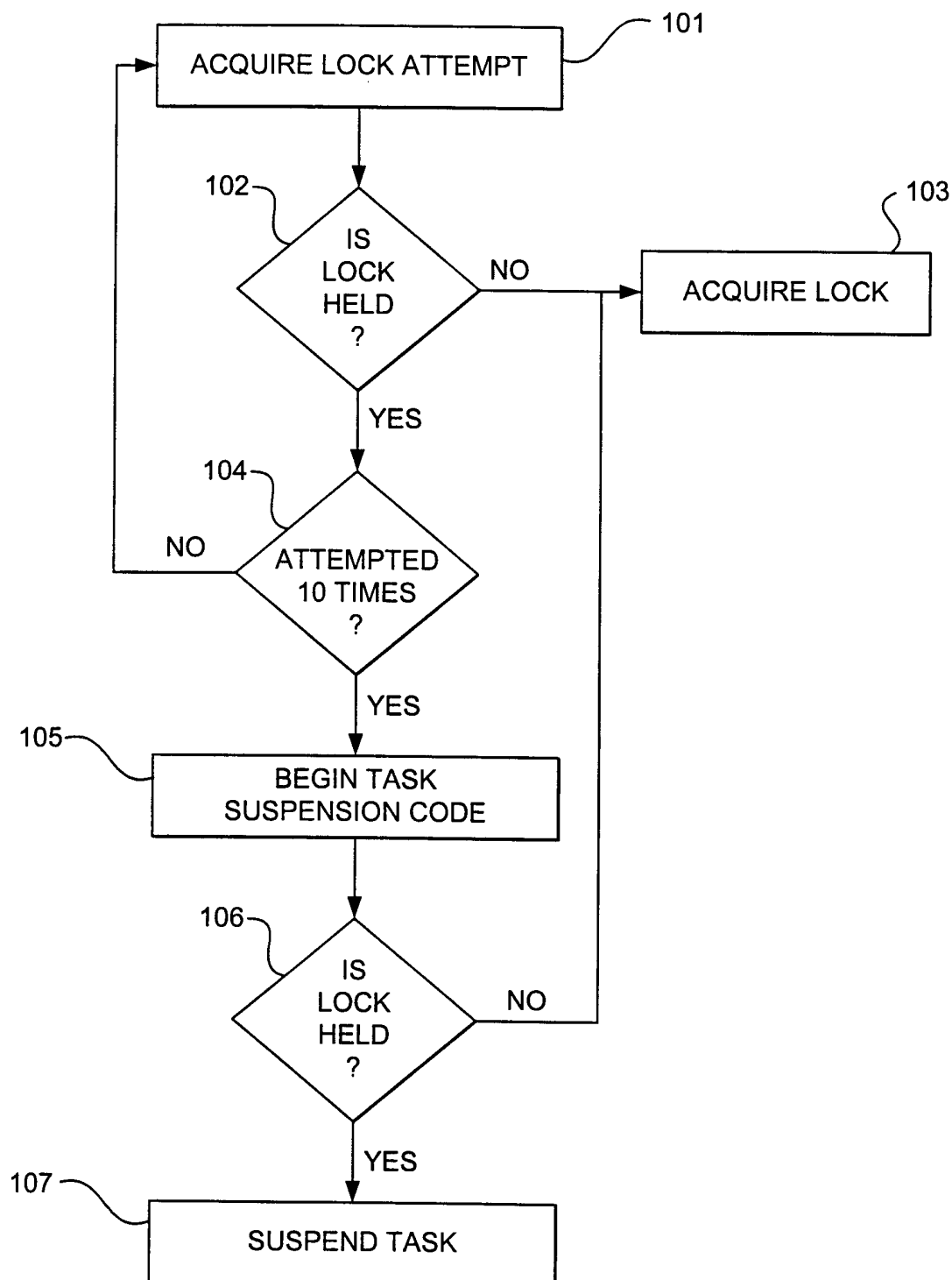


FIG. 1