 Title: SYNCHRONIZATION OF MULTIPLE PROCESSOR CORES

Abstract: The invention relates to a spinlock-based multi-core synchronization technique in a real-time environment, wherein multiple processor cores perform spinning attempts to request a lock and the lock is located at most one of the multiple cores for a mutually exclusive operation thereof. A method embodiment of the technique comprises the steps of allocating the lock to the first core requesting it; establishing for each core an indication of a waiting time for receiving the lock; selecting at least one of the spinning cores based on the waiting time indications; and, upon return of the lock, conditionally allocating the lock to the selected core, if the selected core performs a spinning attempt within a predefined time window starting with the return of the lock.
Synchronization of multiple processor cores

5 Technical Field

The invention relates to a technique for synchronizing an operation of multiple processor cores. More particularly, the invention relates to a spinlock-based multi-core synchronization technique in a real-time environment.

Background

A multi-core computing system comprises multiple processing units or 'cores'; for example, a dual-core CPU comprises two processor cores, a quad-core CPU comprises four cores, etc., wherein the cores may typically be provided on a single chip (a single integrated circuit). A multi-core system allows true parallel processing of multiple processes or tasks such that, e.g., each process or task is executed on a dedicated core. The cores may share common resources, for example a cache or other storage, and/or the interconnections to other portions of the system. The usage of the shared resources has to be properly controlled, i.e. some synchronization of the multiple cores is required in particular to ensure a sequential access to the resources.

Typically a low-level synchronization mechanism is provided by a microcontroller of the multi-core system in the form of a set of instructions; for example, atomic test_and_set instruction(s) may be used to implement a spinlock mechanism. This low-level mechanism may then serve as the basis for service functions (e.g., a semaphore functionality) provided by an operating system.

A spinlock mechanism realizes a mutually exclusive operation of multiple cores with regard to a shared resource. A simple spinlock mechanism may work as follows:

- multiple cores request the lock;
- the lock is allocated to exactly one of the cores; this core may start processing by using the unlocked resource;
- the other cores continue to request the lock, i.e. they repeatedly and continuously check whether the lock is available ("spinning");
- the operating core returns the lock after end of operation;
the lock is allocated to the core which happens to perform the first spinning attempt after the lock has been returned; this core may start processing, etc.

Thus, an essential aspect of the spinlock mechanism is that the lock, if available, is allocated to the first core requesting it. This spinlock mechanism provides for a simple and generally efficient synchronization below the level of operating system real-time scheduling and is therefore widely employed, for example in personal computers or general purpose computers.

However, the simple spinlock mechanism does not allow to observe timing constraints. Specifically, it is not possible to calculate the maximum time or a time boundary it may take to fulfil a task. The simple spinlock mechanism allows a "starvation" of cores; i.e. it may happen that a core never receives the lock, because other cores are always faster. Thus, the spinlock mechanism is not deterministic.

Moreover, a fair scheduling should also take into account the priorities of the tasks to be executed on the cores. The simple spinlock mechanism allows "priority inversion", i.e. a low priority task executed on one core may block a high priority task on another core which has to wait for the lock.

The above drawbacks make the simple spinlock inappropriate for time-critical environments and in particular for real-time environments, i.e. configurations in which one or more real-time applications are executed on the multi-core system. A typical real-time environment is an automotive environment in which, e.g., safety-critical applications have to be performed. The behaviour of a real-time system (a multi-core system configured for a real-time environment) must be deterministic and fair. In order to make a spinlock-based scheduling applicable for real-time environments, the simple spinlock thus has to be modified.

One solution could be to provide an additional mechanism for killing a task executed on a core once a time limit is exceeded. However, this mechanism requires additional control overhead, for example an additional mechanism to release the lock, and is still neither deterministic nor fair.

According to another solution, the scheduler may assign a ticket to each spinning core. Each ticket comprises a unique number which may be seen as a 'waiting number' such that the spinning cores are ordered by means of their ticket numbers into a strictly sequential sequence. The lock is then allocated to one core after another as
indicated by the tickets. While at first glance this mechanism seems to guarantee that each of the spinning cores will receive the lock, there are still problems. Consider, for example, the case that a waiting core is no longer interested in getting the lock, for example because a time limit for the task to be performed has been exceeded, i.e. the waiting time has been too long and another processing path will be followed instead. When arriving at the ticket of this core, the proceeding controlled by the ticket sequence will hang-up, as the core will never return the lock. Thus, this ticket-based mechanism is not deterministic.

More sophisticated, queue-based techniques may be used, in which the spinning cores are sorted into a queue by defining preceding and succeeding relations between the cores. With regard to the queue then additional mechanisms may be defined, for example, a time-out mechanism which allows that a core may leave the queue due to an exceeded time limit or a re-scheduling from the operating system. However, such queue-based spinlocks are quite complicated. In particular for multi-core systems comprising only few cores the overhead introduced by queue-based spinlocks may be large compared to a simple spinlock. And the overhead will get even larger when additional mechanisms have to be introduced in order to consider the priorities of the tasks running on the cores.

Summary

There is a demand for a fair and efficient technique for synchronizing multiple processor cores.

This demand is satisfied by a method for synchronizing an operation of multiple processor cores performing spinning attempts to request a lock, wherein the lock is allocated to at most one of the multiple cores for a mutually exclusive operation thereof. The method comprises the steps of allocating the lock to the first core requesting it; establishing for each core an indication of a waiting time for receiving the lock; selecting at least one of the spinning cores based on the waiting time indications; and, upon return of the lock, conditionally allocating the lock to the selected core, if the selected core performs a spinning attempt within a predefined time window starting with the return of the lock.
The steps of allocating the lock to the first requesting core and of establishing the waiting time indications may be performed in any sequential order or may be performed in parallel to each other.

The waiting time indications established for the spinning cores may comprise at least one of time stamps and a monotonous sequence of numbers. For example, a ticket may be assigned to each core for which a spinning attempt is detected. A ticket number may be represented by an integer value counted up from one detected spinning attempt to the next.

In one implementation, the waiting time indications are established while the lock is allocated to the first core; for example, a counter for determining the ticket numbers may be reset when the lock is allocated to the first core and may then be counting up with the detected spinning attempts.

In one realization of the method, the step of selecting one of the cores comprises selecting the core with the longest waiting time. This may, for example, be the core with the oldest time stamp or the lowest ticket number. In one mode of this realization, the core with the longest waiting time will get the lock.

A variant of the method comprises the step of assigning a priority to each spinning core. In this variant, the step of selecting a core based on the waiting time indications comprises selecting a core based on its priority. For example, a core may only be selected if its priority is at least the priority of the core to which the lock is currently allocated.

The steps of selecting a core and allocating the lock to the selected core may be repeatedly performed until the time window expires. Additionally or alternatively, the lock may be allocated to the selected core for a predefined time period. In one implementation, the predefined time period is equal to the duration of the time window. The length of one or both of the time window and the time period may allow few instructions only to be performed.

One variant of the method comprises the step of sending a control signal to the core to which the lock has been allocated, if on expiry of the predefined time period the core still holds the lock. The control signal may indicate a request to return the lock.
If within the predefined time window no spinning attempt of the core with the longest waiting time is detected, the lock may be allocated instead to the core with the second longest waiting time (if a spinning attempt corresponding to this core is detected).

The abovementioned demand is moreover satisfied by a computer program product, which comprises program code portions for performing the steps of one or more of the methods and method aspects described herein when the computer program product is executed on one or more computing devices, for example a multi-core system. The computer program product may be stored on a computer readable recording medium, such as a permanent or re-writeable memory within or associated with a computing device or a removable CD-ROM, DVD or USB-stick. Additionally or alternatively, the computer program product may be provided for download to a computing device, for example via a data network such as the Internet or a communication line such as a telephone line or wireless link.

Further, the abovementioned demand is satisfied by a spinlock module for synchronizing multiple cores. The spinlock module comprises a component adapted to allocate the lock to the first core requesting it; a component adapted to establish for each core an indication of a waiting time for receiving the lock; a component adapted to select at least one of the spinning cores based on the waiting time indications; and a component adapted to conditionally allocate, upon return of the lock, the lock to the selected core, if the selected core performs a spinning attempt within a predefined time window starting with the return of the lock.

The spinlock module may further comprise a component adapted to assign a priority to each spinning core.

The abovementioned demand is also satisfied by a multi-core system comprising a spinlock module as outlined above.

**Brief Description of the Drawings**

In the following, the invention will further be described with reference to exemplary embodiments illustrated in the figures, in which:

Fig. 1 schematically illustrates an embodiment of a spinlock module;
Fig. 2 is a flow diagram illustrating an operation of the spinlock module of Fig. 1; and

Fig. 3 illustrates a control table implemented in the spinlock module of Fig. 1.

**Detailed Description of Preferred Embodiments**

In the following description, for purposes of explanation and not limitation, specific details are set forth, such as a particularly configured spinlock scheduler, in order to provide a thorough understanding of the current invention. It will be apparent to one skilled in the art that the current invention may be practiced in other embodiments that depart from these specific aspects. For example, for reasons of a clear functional illustration, below it is described that a lock is realized as a lock status register which is accessed only by the scheduler. In other embodiments, additionally or alternatively the cores or processes may directly access a register holding the lock by using, e.g., atomic instructions provided for that purpose. As a further example, while the embodiments below include selecting cores based on their priority, in other embodiments no priority handling may be implemented at all. The techniques described herein may generally be used for synchronizing multiple cores, but may also be used for synchronizing multiple processes on a system with a single processor core.

Those skilled in the art will further appreciate that functions explained herein below may for example be implemented in the form of a hardware circuit, a combination of software and hardware, or on a pure software (firmware) basis. It will be appreciated that when the current invention is described as a method, this method may be embodied in a processor, e.g. a microcontroller, and a memory coupled to the processor, wherein the memory is encoded with one or more programs that perform the methods disclosed herein when executed by the processor.

Fig. 1 schematically illustrates an embodiment of a multi-core system 100 comprising multiple cores C1, C2, C3 and a spinlock module (or scheduler) 102. The spinlock module 102 comprises a history register 104 and a lock state register 106. The history register (or field) 104 may for example be a 1-byte or 2-byte register, while the lock state register may be a 1-bit register. The spinlock module 102 further comprises a signal input component 108, a ticket assignment component 110, a spinning
detection component 112, a lock allocation component 114 and a lock state change
component 116.

An operation of the spinlock module 102 and an interworking of the components
thereof will be described with reference to the sequence of operational steps illus-
trated in the flow diagram of Fig. 2. Generally, the spinlock module 102 operates to
synchronize the multiple processor cores Cl, ..., C3 with regard to a usage of a com-
monly shared resource (not shown in Fig. 1). The spinlock module 102 has to ensure
a strict sequential ordering of the access operations of the multiple cores to the
shared resource. To this end, the spinlock module or scheduler 102 allocates a lock
represented by the lock state register 106 at any given time to at most one of the
multiple cores Cl, ..., C3 for a mutually exclusive operation thereof. The information
stored in both the history register 104 and the lock state register 106 forms the basis
for synchronizing the multiple cores.

Referring to Hg. 2, in step 202, the spinlock module 102 operates to establish an
indication of a waiting time for receiving the lock for each of the multiple cores. More
specifically, the cores Cl, ..., C3 may generally be adapted to request a ticket before
attempting to receive the lock. For exemplary purposes, in Fig. 1 it is assumed that
core Cl sends a signal 118 to the spinlock module 102. The signal 118 comprises an
indication that an assignment of a ticket and a priority is requested by the core Cl.

In the embodiment illustrated in Fig. 1, any signal from the cores is received in the
signal input component 108, which is configured to forward the signal or generate an
internal trigger to an appropriate one of the further components of the spinlock
module 102. On reception of the signal 118, the component 108 accordingly triggers
the ticket assignment component 110, which operates to assign a ticket and a priority
to the requesting core Cl. For this purpose, a sub-component 119 for generating
a ticket number is driven by the ticket assignment component 110. The sub-
component 118 may be a simple counter which upon being triggered by the compo-
nent 110 generates an integer number by adding +1 to the previously generated
number, and provides the generated number to the component 110. In another
embodiment, the sub-component 119 may generate a timestamp from a system
time, for example by accessing a system clock. Any other mechanism for generating
a ticket number may be employed, as long as the generated ticket numbers allow a
sequential ordering of the multiple cores reflecting the sequence in time of their
ticket requests. For example, besides monotonously increasing sequence of numbers,
also a monotonously decreasing sequence of numbers may be used. The term 'numbers' may not only refer to integer numbers, but also to binary numbers, hexadecimal numbers, or similar numerical representations.

The ticket assignment component 110 also triggers a priority sub-component 120 which determines a priority to be assigned to the requesting core CI. The sub-component 120 may determine the priority within the context of the requesting core, i.e. may access additional data related to the requesting core, such as data indicating a priority or importance of the one or more tasks to be executed on the requesting core. The ticket assignment component 110 may trigger provision of a ticket comprising the generated ticket number and priority to the requesting core CI, or the ticket may otherwise be assigned to this core, for example by storing the ticket in association with an ID-number of the requesting core in a buffer (not shown in Fig. 1). In general, any assignment mechanism may be employed which allows the scheduler 102 to associate generated tickets to the spinning cores later on.

In step 204, the spinlock module 102 operates to allocate the lock to the first core requesting it. More specifically, and with regard to the configuration illustrated in Fig. 1, it is exemplarily assumed that the core C2 sends a signal 121 to the spinlock module 102 indicating a request for the lock; i.e. the signal 121 is a spinning attempt performed by core C2. The signal 121 may be identified as a spinning attempt by the signal input component 108, e.g., because a ticket comprising a ticket number and priority is indicated in the signal 121. The ticket may have been assigned by the core C2 in a similar way as described above for the example of core CI. The ticket may be included in the signal 121 or may be stored in association with the spinlock module 102 and may be identified based on an ID-number for the core C2 included in the signal 121, for example.

The signal input component 108 triggers the spinning detection component 112 by indicating the core C2 and/or the ticket assigned to this core. The component 112 normally operates to write the ticket number and priority of the ticket assigned to core C2 into the history register 104. It is assumed that before the writing access the history register is in an initial state, which may be defined by a particular number such as O0001 or any other predefined state, which is generally referred to as the INIT state hereinafter.
The lock allocation component 114 generally operates to allocate the lock to a spinning core in case the lock state represented in the lock state register 106 indicates that the lock is in status 'FREE' (as opposed to the status 'LOCKED'). The lock allocation component 114 may regularly poll the register 106 in order to determine whether an allocation operation has to be performed. Additionally or alternatively, an operation of the lock allocation component 114 may be triggered by control signals from other components. For example, in the course of processing the spinning attempt 121 of core C2 described above, the spinning detection component 112 may trigger the lock allocation component 114. Fig. 1 illustrates a general signal line 122 between the components 112 and 114, which may be used for this purpose. Any other mechanism which lets the lock allocation component 114 detect a change in the lock state register 106 may also be used.

Operations O1 - O4 performed by the spinlock module 102, in particular the components 112 and 114, in response to a status or status change of the history register 104 and lock state register 106, are exemplarily illustrated in Fig. 3. In the embodiment of Fig. 1, it is assumed that a control table representing the content of Fig. 3 is provided for access by the components 112 and 114 in a buffer 124, such that the control table 300 may define an operation of these components.

Referring specifically to step 204 in Fig. 2, the control table 300 in Fig. 3 prescribes the operation O1 to be performed by the lock allocation component 114 in case the status of the history register 104 changes from 'INFT' to holding a ticket and the lock state is 'FREE'. In this case, the operation O1 comprises to allocate the lock to the core corresponding to the ticket currently stored in the history 104. Taking the above discussed example further, this is the ticket of core C2 stored by the component 112 in the history 104 in response to the spinning attempt 121. Based on the instruction O1, the lock allocation component 114 extracts the ticket number stored in the history 104 and allocates the lock to the corresponding core C2 (arrow 126 in Fig. 1). As the spinning attempt 121 is assumed to be the first spinning attempt after the history has been (re-)set into the state 'INIT', the operation O1 specifies that the lock is to be allocated to the first core requesting it.

In parallel to allocating the lock to core C2, the lock allocation component 114 may signal the lock state change component 116 to change the lock state in register 106 from 'FREE' to 'LOCKED'.
After the lock has been allocated to the first requesting core in step 204, in step 206 the spinlock module 102 operates to select at least one of the spinning cores based on their respective waiting time indications, i.e. tickets. More specifically, the cores which do not have received the lock, i.e. cores C1 and C3 in Fig. 1, will continue spinning for the lock (this is indicated by short arrows 128 in Fig. 1 for sake of illustration). For each spinning attempt forwarded by the signal input component 108, the spinning detection component 112 analyses the assigned ticket (operation 0.2 in control table 300 in Fig. 3). The ticket (ticket number and priority) is written to the history register 104 in case two conditions are fulfilled:

1. The priority assigned to the spinning core has to be equal to or higher than the priority currently buffered in the history 104. If this is not the case, the history 104 is not changed and the detection component 112 stops operation.

2. The ticket number assigned to the spinning core has to be an earlier value than the ticket number currently stored in the history 104. For example, in case the ticket numbers are generated from a monotonously increasing number sequence, the spinning core has to have assigned a number which is smaller than the ticket number currently stored in the history 104. A first ticket will be referred to as 'earlier' or 'older' than a second ticket in case the first ticket has been assigned to the corresponding core earlier or before the second ticket has been assigned to its core, irrespective of whether the ticket numbers are based on time stamps, a clock counter, increasing/decreasing sequence of numbers, etc.

In case the ticket number and priority of a spinning core does not fulfil one or both of these conditions, the history 104 is not changed. In this way, the ticket for a particular core will be selected, i.e. in the history the ticket assigned to a core ready to execute a high priority task and waiting already for a long time to get the lock will be stored. For later reference, we may assume that core C3 is selected, i.e. the ticket assigned to core C3 is stored in the history 104.

In step 208, upon return of the lock from the first core, the lock is conditionally allocated to the core selected in step 206, if the selected core performs a spinning attempt within a predefined time window, which starts with the return of the lock. More specifically and with reference to the exemplary embodiment illustrated in Fig. 1, the core C2 which has been allocated the core by signal 126 may, upon completion of its task, provide a signal 129 indicating a lock return to the spinlock module 102. The signal input component 108 forwards the return signal to the lock state
change component 116, which operates in response to the forwarded signal to change the lock state held in register 106 from 'LOCKED' to 'FREE'.

Triggered by the status change, the spinning detection component 112 and lock allocation component 116 change its mode of operation during a predefined time window $\Delta t_1$, wherein the value of $\Delta t_i$, i.e. the duration of the time window, is stored in a storage component 130 for access by the spinning detection component 112 and lock allocation component 114. The spinning detection component 112 stops the operation 0.2 of evaluating and conditionally writing tickets to the history register 104. Instead, the component 112 performs operation 0.3 (see Fig. 3) which includes writing the ticket number of any spinning core to a current register 132, which may, e.g., be a 1-byte register (the priority of the spinning core need not to be stored in the current register 132).

Within the time window $\Delta t_i$, the lock allocation component 114 operates to compare the ticket number buffered in the current register 132 to the ticket number stored in the history register 104 (depicted as operation 0.4 in the control table 300 illustrated in Fig. 3). The lock allocation component 114 may, for example, regularly poll the current register 132, or may be triggered via signal line 122 by the spinning detection component 112. In case it turns out that a ticket number stored in the current register 132 equals the ticket number stored in the history register 104, this means that a spinning attempt of the core which has been selected as harbouring a high-priority task and waiting for a long time has been detected. In this case, the lock allocation component 114 allocates the lock to this core, i.e. the core corresponding to the ticket number stored in the history register 104 and current register 132. In the example discussed here with reference to the configuration depicted in Fig. 1, the selected core C3 has performed a further spinning attempt during the time window $\Delta t_i$ and therefore the lock is allocated to the selected core C3. Further, the component 114 triggers the lock state change component 116 to set the lock state to 'LOCKED'.

Instead of the current register 132, any other configuration might also be employed which allows to compare the ticket number stored in the history 104 with the ticket number assigned to a currently spinning core.

For the case the lock is allocated to the selected core during the time window $\Delta t_i$, in the embodiment described here, a value of a predetermined time period $\Delta t_2$ is stored.
in the configuration parameter storage 130. The time period $\Delta t_2$ prescribes a maximum operation time for the selected core before to return the lock. In case the selected core does not return the lock before the expiry of the time period $\Delta t_2$, the lock allocation component 114 or any other component of the spinlock module 102 sends a control signal to the selected core in order to force a return of the lock. In the example of Fig. 1, this control signal is illustrated as a signal 134 sent to the selected core C3. Provision of the time period $\Delta t_2$ is optional: In other embodiments, no such additional parameter may be provided, e.g. in an environment in which it is clear that selected cores will return the lock after few operation cycles.

In case the selected core returns the lock before the waiting time window $\Delta t_i$ has been elapsed, the step 208 may be repeated until the time window $\Delta t_i$ expires. For such a situation, the step 206 may comprise selecting two (or more) tickets of spinning cores with a high priority and early ticket numbers. The history register 104 may be adapted to store the selected two (or more) tickets. In another embodiment, the selecting step 206 is repeated during the time the lock is allocated to the selected core within the time window $\Delta t_i$. Upon return of the lock within the time window $\Delta t_i$, the lock may then again be allocated to the core whose ticket is stored in the history register 104.

In step 210, whether or not the lock has been allocated to the selected core(s), the history 104 is reset into the state 'NFT in any case after the time window $\Delta t_i$ has been elapsed. The operation of the spinlock module 102 may then continue with steps 202 and 204, as described above.

The time window $\Delta t_i$ and time period $\Delta t_2$ may be configurable and may be defined independently of each other, i.e., depending on the concrete environment $\Delta t_i$ may be chosen smaller or larger than $\Delta t_2$. Typically, $\Delta t_i$ may be chosen such that only a small number of operations can be performed in this time. In some embodiments, $\Delta t_i = \Delta t_2$. In these embodiments, only a single parameter $\Delta t$ may be provided for the time window, during which the scheduler waits for another spinning attempt of the selected core, and the time period for which the lock is at most allocated to the selected core(s).

The steps 202 and 204 in Fig. 2 may be performed in any order. In particular, the step 202 of establishing tickets for spinning cores will be performed once for each
core, i.e. may generally be performed repeatedly and continuously when the lock is allocated to a core.

While it has been described above that in the history register 104 the tickets are directly stored, in other embodiments a reference, pointer or similar link to the ticket stored elsewhere may be held in the history register.

While for the sake of illustration the proposed multi-core synchronization technique is applied to a multi-core system with three cores in the embodiment described above, it is to be understood that the technique may be applied to multi-core systems with any number of cores, for example to systems with only two cores and systems with more than three cores, e.g., four or five cores, and also to systems with a large number of cores, e.g., 32, 64 or 128 core systems.

Provision of a history register allows buffering the ticket number or a similar waiting time indication for a core to which one wants the lock to be allocated in order to guarantee a deterministic and fair behaviour of the spinlock mechanism. The core selected in this way has the chance to gain the lock during a small time window, during which the spinlock module waits for a spinning attempt of the selected core, i.e. the core whose ticket is buffered in the history 104. This mechanism allows considering long waiting and/or high priority cores to gain the lock, and thus leads to a deterministic and fair behaviour. No priority inversion does occur.

As the core is allocated to the selected lock only in case this core still spins around, no blocking situation can occur in case the selected core has meanwhile stopped spinning, e.g. because it has been terminated or a time limit has exceeded and the core meanwhile tries another processing path. A starvation of cores does not occur and no extra release mechanism for a blocked lock situation needs to be implemented.

The provision of the time window for allocating the lock to the selected core will not lead to a decrease in processing efficiency, as in many typical cases the oldest / high-priority core will still be spinning and thus the lock will be allocated to this core rapidly after the start of the time window.

The techniques proposed herein increase the usability of multi-core systems as the hardware basis for supporting multiple applications. In particular, it can be ensured
that the execution of safety-critical applications cannot be blocked by other applications or tasks. A multi-core system implementing the techniques illustrated herein may be employed in a real-time environment, for example in an automotive environment which uses multi-core microcontrollers.

While the current invention has been described in relation to its preferred embodiments, it is to be understood that this description is for illustrative purposes only. [9ii] Accordingly, it is intended that the invention be limited only by the scope of the claims appended hereto.
Claims

1. A method for synchronizing multiple processor cores performing spinning attempts to request a lock \([M_2]\), wherein the lock is allocated to at most one of the multiple cores (Cl - C3) for a mutually exclusive operation thereof, the method comprising the steps of:
   - allocating (204) the lock (106) to the first core (C2) requesting it;
   - establishing (202) for each core (Cl - C3) an indication of a waiting time for receiving the lock (106);
   - selecting (206) at least one of the spinning cores (C3) based on the waiting time indications; and,
   - upon return of the lock (106), conditionally allocating (208) the lock to the selected core (C3), if the selected core performs a spinning attempt within a predefined time window (At_1) starting with the return of the lock.

2. The method according to claim 1,
   wherein the step of selecting (206) one of the cores comprises selecting the core (C3) with the longest waiting time.

3. The method according to claim 1 or 2,
   wherein the waiting time indications established for the spinning cores (Cl - C3) comprise at least one of time stamps and a monotonous sequence of numbers.

4. The method according to any one of the preceding claims,
   comprising the step of assigning a priority to each spinning core, wherein the step of selecting a core based on the waiting time indications comprises selecting a core based on its priority.

5. The method according to claim 4,
   wherein a core is selected if its priority is at least the priority of the core to which the lock is currently allocated.
6. The method according to any one of the preceding claims, wherein the steps of selecting (206) a core and allocating (208) the lock to the selected core is repeatedly performed until the time window (At₁) expires.

7. The method according to any one of the preceding claims, wherein the lock is allocated to the selected core for a predefined time period (Δt₂).

8. The method according to claim 7, comprising the step of sending a control signal (134) to the core to which the lock has been allocated, if on expiry of the predefined time period (Δt₂) the core still holds the lock.

9. The method according to claim 7 or 8, wherein the predefined time period (Δt₂) is equal to the duration of the time window (At₁).

10. The method according to any one of claims 2 to 9, comprising the step of allocating the lock to the core with the second longest waiting time, if within the predefined time window no spinning attempt of the core with the longest waiting time is detected.

11. A computer program product comprising program code portions for performing the method of any one of the preceding claims when the computer program product is executed on one or more computing devices.

12. The computer program product of claim 11, stored on a computer readable recording medium.

13. A spinlock module (102) for synchronizing multiple cores (C₁ - C₃), comprising:
   - a component (114, 124) adapted to allocate the lock (106) to the first core requesting it;
   - a component (110, 119) adapted to establish for each core an indication of a waiting time for receiving the lock;
   - a component (112, 104, 124, 132) adapted to select at least one of the spinning cores based on the waiting time indications; and
   - a component (114, 124, 104, 132) adapted to conditionally allocate, upon turn of the lock, the lock to the selected core, if the selected core performs a
spinning attempt within a predefined time window \((At_1)\) starting with the return of the lock.

14. The spinlock module according to claim 13, further comprising a component \((110, 120)\) adapted to assign a priority to each spinning core.

15. A multi-core system \((100)\) comprising a spinlock module \((102)\) according to claim 13 or 14.
Synchronization of multiple processor cores

202
Establish for each core an indication of a waiting time for receiving the lock

204
Allocate the lock to the first core requesting it

206
Select at least one of the spinning cores based on the waiting time indications

208
Upon return of the lock, conditionally allocate the lock to the selected core, if the selected core performs a spinning attempt within a predefined time window starting with the return of the lock

210
Reset history into initial state

Fig. 2
<table>
<thead>
<tr>
<th></th>
<th>History</th>
<th>Lock state</th>
<th>Time window</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>init -&gt; ticket</td>
<td>FREE</td>
<td>-</td>
<td>Allocate to first requesting core</td>
</tr>
<tr>
<td>O2</td>
<td>ticket -&gt; ticket</td>
<td>LOCKED</td>
<td>-</td>
<td>Conditionally write to History</td>
</tr>
<tr>
<td>O3</td>
<td>ticket -&gt; ticket</td>
<td>FREE</td>
<td>+</td>
<td>Write to Current</td>
</tr>
<tr>
<td>O4</td>
<td>ticket -&gt; ticket</td>
<td>FREE</td>
<td>+</td>
<td>Allocate to selected core when in Current</td>
</tr>
</tbody>
</table>

Fig. 3
A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F9/46

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

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

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, IBM-TDB, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>CRAIG T S ED - INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS: &quot;QUENING SPIN LOCK ALGORITHMS TO SUPPORT TIMING PREDICTABILITY&quot; PROCEEDINGS OF THE REAL TIME SYSTEMS SYMPOSIUM. RALEIGH-DURHAM, DEC. 1 19931201 LOS ALAMITOS, IEEE COMP. SOC. PRESS, US, vol. -, 1 December 1993 (1993-12-01), pages 148-157, XP000457643 abstract page 149, left-hand column, paragraph 1 - paragraph 5 page 151, right-hand column, paragraph 2 - page 152, left-hand column, paragraph 1 page 153, left-hand column, paragraph 1 - paragraph 5 page 154, right-hand column., paragraph 3 - page 155, left-hand column, paragraph 1</td>
<td>1-15</td>
</tr>
<tr>
<td>X</td>
<td>Further documents are listed in the continuation of Box C</td>
<td>See patent family annex</td>
</tr>
</tbody>
</table>

D. Additional information

'S' special categories of cited documents

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

'E' earlier document but published on or after the international filing date

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

'O' document referred to in an oral disclosure, use, exhibition or other means

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

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

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

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

'Z' document member of the same patent family

Date of the actual completion of the international search: 5 February 2009

Date of mailing of the international search report: 18/02/2009

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

Authorized officer: Milasinovic, Goran
**C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT**

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>HIROAKI TAKADA ET AL: &quot;QUEUING SPIN LOCK ALGORITHMS WITH PREEMPTION&quot;</td>
<td>1-15</td>
</tr>
<tr>
<td></td>
<td>ISSN: 0882-1666 abstract page 15, right-hand column, paragraph 2 - page 16, left-hand column, paragraph 2 page 16, right-hand column, paragraph 3 - paragraph 4 page 17, left-hand column, paragraph 4 - right-hand column, paragraph 4 page 18, right-hand column, paragraph 1</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>TAKADA H ET AL: &quot;Predictable spin lock algorithms with preemption&quot;</td>
<td>1-15</td>
</tr>
<tr>
<td></td>
<td>ISBN: 978-0-8186-5710-8 abstract page 2, left-hand column, paragraph 2 - right-hand column, paragraph 4 page 3, left-hand column, section 3, paragraph 1 - right-hand column, paragraph 3</td>
<td></td>
</tr>
</tbody>
</table>