

(51) International Patent Classification: G06F 09/00
(54) Title: METHOD AND APPARATUS TO USE TASK PRIORITY TO SCALE PROCESSOR PERFORMANCE

(21) International Application Number: PCT/US02/17309

(22) International Filing Date: 3 June 2002 (03.06.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

09/873,591 4 June 2001 (04.06.2001) US

(71) Applicant (for all designated States except US):
RADYSYS CORPORATION [US/US]; 5445 NE Dawson Creek Drive, Hillsboro, OR 97124 (US).

(72) Inventors; and


[Continued on next page]

(57) Abstract: The invention uses a general-purpose computer with a processor capable of operating at a plurality of performance levels, and has an operating system with the capability to set a plurality of task priority levels for task performed on the computer. The method reads a task's priority level, associates the task's priority level with a performance level, and sets the processor to operate at the performance level.
Published:
— with international search 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.
METHOD AND APPARATUS TO USE TASK
PRIORITY TO SCALE PROCESSOR PERFORMANCE

Background of the Invention

1. Field of the Invention.

The present invention relates to a method for using task priority to scale processor performance. In particular, to a method and apparatus that reads a tasks priority level and associates that level with a processor performance level, and then sets the processor to that performance level.

2. Background.

For a variety of reasons some modern computer systems include mechanisms to alter the speed at which the core processor operates. Recent examples of variable performance products include Intel’s XScale architecture, and processors made by Transmeta. For example, systems will vary processor performance in order to manage processor heat build up. By slowing processor speed the system can dissipate excess heat. In this manner the system manages the trade off between processor performance and power consumption to avoid overheating. In systems that utilize batteries for operation, like laptop computers and the like, similar trade offs exist. In order to extend or conserve battery power the system adjusts the performance of the processor. Higher performance correlates to greater power consumption, and therefore, less operating time on a single battery.

Computer systems utilize a variety of techniques to alter or adjust processor performance. One such means consists of adjusting the processor’s internal clock speed. This directly affects performance in that processors generally execute instructions based on the
clock rate. Other systems increase the power savings by regulating voltage with the clock rate. At lower clock rates the processor can operate correctly at a lower voltage.

A previously unrelated feature of modern computer system comprises the concept of task priority. A task is an operating system concept whereby a concurrent thread of control is recognized and controlled by the operating system. The operating system associates certain information with the task to coordinate the system to accomplish the task. Whenever a computer system executes a set of instructions designed to accomplish a certain job or task, the operating system maintains certain information related to the task for bookkeeping purposes. Thus, as tasks change the operating system constantly must maintain task information to keep track of the various tasks being performed by the computer system. The process of swapping between tasks is often called context switching. One of the pieces of information that the operating systems tracks for a particular task consists of the task priority. In order to facilitate orderly processing, tasks receive priorities that help the operating system determine the relative importance of the tasks that compete for processor time. On most operating systems the more important the task the higher the task priority, and the less important the task the lower the priority. That ordering is assumed, reversing it such that lower priorities denote more urgent tasks would entail obvious changes to the methods detailed herein. The operating system will process tasks in accord with their priority. Of course, the number of tasks varies from operating system to operating system, from a dozen or less to several thousand.

While processor performance and task priority to some degree both impact the functioning of a computer devices, heretofore there has been no mechanism or motivation to link or associate these concepts in a way to improve operation of a computer device.
Accordingly, a need exists to use task priority to scale processor performance in a computer device.

**Summary of the Invention**

An object of the present invention comprises using task priority to scale processor performance in a computer device.

These and other objects of the present invention will become apparent to those skilled in the art upon reference to the following specification, drawings, and claims.

The present invention intends to overcome the difficulties encountered heretofore. To that end, the invention uses a general-purpose computer with a processor capable of operating at a plurality of performance levels, and has an operating system with the capability to set a plurality of task priority levels for tasks performed on the computer. The method reads a task’s priority level, associates the task’s priority level with a performance level, and sets the processor to operate at the performance level.

**Brief Description of the Drawings**

Figure 1 depicts a flow for the operation of an operating system API.

Figure 2 depicts a flow chart for setting processor performance based on a map value.

Figure 3 depicts a flow chart for context switching.

Figure 4 depicts a flow chart for a driver call for setting processor performance level.

Figure 5 depicts a flow chart for a driver call for converting unsigned integer map values into hardware specific performance values.
Detailed Description of the Invention

In general terms, the present invention consists of four interrelated components: (1) an operating system application program interface (API) that initially maps priority and performance; (2) code for setting a task's processor performance level based on the map; (3) altered context switching code to account for task performance level and (4) a performance control driver to convert the map into hardware specific settings and to set performance levels.

In particular, Figure 1 shows a flow chart depicting the operation of the operating system API. The API would map task priority into performance setting based on an algorithm, either pre-defined or in the alternative a dynamic algorithm could be used. For example, the API might contain an initial level and an increment. All task priorities below the initial level would receive the lowest possible performance level. Each progressively higher task priority would receive an incrementally higher performance level, up to the maximum possible performance level at which point all higher priority tasks would receive the maximum performance level. Alternatively, the highest task priority could receive the highest performance level, and all lower task priorities would be receive a correspondingly decremented performance level until the lowest possible performance level was reached.

With operating systems with a large number of task priorities, like Microwave Systems Corporation’s OS–9 which has tens of thousands of task priorities, a broad range of task priorities would map to a single performance level. In other words, in some systems a one-to-one mapping may not be possible. Those of ordinary skill in the art will appreciate the fact
that the specific algorithm for mapping task priority to performance level can and will vary without departing from the scope of the present invention.

As shown in Figure 1, the API would first check to determine if a map is specified. In other words, the routine inquires into whether the system has the information necessary to make a map. If not, the API returns an error. Next, the API would conduct an integrity check to ensure that if a map is specified that the map contains valid values. In other words, the API needs to determine if the existing task priority values and the performance levels are the correct values and levels for the particular system. If not, the API returns an error. At this point the API applies one of the foregoing algorithms to map the task priority settings to performance levels. Preferably, the API creates a map by calculating an unsigned integer value representing a performance level corresponding to each task priority. The API creates a map independent of the specific hardware components of the system. Thus, at this point the performance values, while meaningful in relation to each other, lack relevance to the specific system hardware.

The API, therefore, needs to call a performance control driver to convert the performance map into specific system settings corresponding to actual performance values. For example, if processor performance varies according to voltage the performance control driver would convert the unsigned integers into a voltage, or if the performance varies according to clock speed the performance control driver would convert each unsigned integer into a clock speed. In other words, the performance control driver converts the generic unsigned integers into specific performance values with meaning to the hardware components of the particular system. Of course, the step is merely added to the pre-existing routine used
to calculate the specific performance control setting, since the system already can adjust performance. Next, the newly created map is stored to memory for later recall.

Of course, it is not essential to call the performance control driver during the API setup stage, however, it will almost certainly prove more efficient. In the alternative, the system could store the map with the relative unsigned integer performance values passing the generic performance values to the performance control driver each time that the system performs a context switch. The performance control driver would then need to convert the generic number into a device specific value for each context switch. If the performance control driver completes the device specific conversion initially, for example at boot time, then the performance control driver could be supplied with the device specific setting at each context switch, thereby saving time and processor resources.

Returning to the second component of the present invention, namely, the code for setting processor performance based on the map. Figure 2 depicts a flow chart for this portion of the invention. Each task has a task descriptor to store such things as the task’s priority level. It is necessary to associate the task’s priority level with a performance level and store that performance level in the task’s descriptor. This routine must be performed initially to update the task descriptor to reflect the performance level with the task’s priority, and thereafter each time a task’s priority is changed to update the descriptor as necessary. For example, a task's priority/performance level can change based on normal processing or in association with a dynamic algorithm for associating task priority with performance. In other words, task priority is not usually a fixed attribute of a task (discussed in greater detail hereinbelow). In any event, normally a change in task priority will require a change to the task descriptor performance level. To set or alter the performance level, the routine begins by
determining whether the new task priority is equal to the old priority. If they are the same then nothing needs to change, otherwise the routine continues. To continue, the routine recovers the stored map and identifies the performance value associated with the new task priority. Then the task descriptor is updated with the new performance value. The routine must then determine if the actual processor performance level matches the new performance value. If not then the system forces a context switch in the manner described hereinbelow in reference to the third component of the invention.

Returning to the third component of the present invention, namely, the altered context switching code. Figure 3 depicts a portion of the context switch flow chart. At some convenient point in the context switch the system must call the performance control driver to change the current processor performance according to the value specified for the task about to become current. The preferred embodiment changes the processor performance as early as possible in the context switch. The first step involves retrieving the performance values from the task descriptor, preferably the task descriptor contains a device specific value. (Alternatively, the context switching code could use the stored map to find the hardware specific performance level corresponding to the task's priority. This step would be accomplished by associating the task's priority level with performance level stored in the map.) Next, the performance control driver is called and passed the performance values retrieved from the previous step. At this point the performance control driver accomplishes the adjustment to the processor. Finally, the context switch code determines if the performance control driver succeed in changing the processor performance level. If not, then an error is logged. After the performance management step is complete, the context switch is completed as normal.
Returning to the fourth component of the present invention, namely, the operation of the performance control driver that converts the map into hardware specific settings and actually set the processor performance level. Figures 4-5 depict flow charts for the two driver calls. Exemplary function prototypes would be of the following form:

```c
error_code Set_Performance (const performance_val * const new_value)
error_code Convert_Mapping_Table (unsigned int in_table[], performance_val * out_table[])
```

The Set_Performance function is used during context switching to actually change the processor performance level based on the specific hardware configuration of the relevant computer system. This begins by first determining if the new performance level is the same as the current performance level. If they are the same then the routine ends, and no change is necessary. Otherwise, the routine proceeds by checking the new value to determine if it is a valid setting. If the new value is not a valid supported setting the routine returns an error. Otherwise, the routine continues and actually performs the hardware manipulation to adjust the performance level to the new value. Again, the actual particulars of this adjustment will vary based on the power management provisions of the system. In some systems processor performance may vary through changes to the clock speed, on others through changes to voltage. Regardless of the exact form of the performance manipulation, the means for changing processor performance should already be known due to the fact that the processor/system is already designed for such manipulation. Next, the routine verifies that the adjustment in processor performance succeeded. If not an error is returned. Otherwise, the new processor performance value is saved for subsequent use and the routine successfully terminates.
The Convert_Mapping_Table function is used to initially convert the generic unsigned integer API map values into hardware specific performance values. The routine begins by executing a loop that checks the validity of all of the entries in the in_table to ensure that there are no unsupported values in the table. The in_table contains the unsigned integer values created during the initial mapping step. The routine returns an error if in_table contains any invalid entries. Otherwise, the routine allocates memory for each instance of the performance_val structure (one instance per entry in the in_table). Next, the routine loops through the in_table and converts each unsigned integer into a hardware dependent value, and stores in each corresponding entry in out_table a pointer to a performance_val structure that stores a specific performance value corresponding to the generic performance value in corresponding entry in in_table. Thus, there is a one-to-one correspondence between the entries in in_table and out_table, through the pointers in out_table.

The present invention is primarily applicable to any multi-tasking software running on a system that supports a processor with variable performance capability. While the most suitable software matrix for the present invention is an extensible operating system, the invention can be applied to any software system that has a notion of multiple priorities. It should be understood that while the present invention is described in terms of systems that assign task priority in a manner wherein the highest numerical priority equates with the highest level of importance, the invention applies equally to systems that assign the highest level of importance to the lowest numerical priority. Additionally, while execution and control of the routines disclosed herein is discussed in reference to computer systems with an operating system, the invention is not limited; any computer system with some form of a run-time environment can utilize the routines disclosed herein.
As mentioned previously, the most suitable system hardware is one that provides a reliable and efficient means for varying processor performance. The best example comprises a processor with architecturally defined mechanism for changing performance, like changing clock speed, which acts quickly and consistently. The present invention can also work with systems with performance-control hardware that, for example, alter processor performance through board-level manipulation of the processor or system voltage to affect processor performance. Performance control in these systems typically is less reliable and slower. To some degree this may limit the effectiveness of the present invention. For example, the longer it takes to adjust and stabilize the processor performance level, the less frequently processor performance should be adjusted. This drawback would not, however, render the invention useless.

Additionally, the changes to processor performance would not be strictly limited to changes in task priority. For example, the present invention could be used to change processor performance in response to exception processing or in response to system interrupts. In the case of interrupts, instead of just letting the system run at the current performance level, the mechanism of the present invention could easily be applied to raise performance levels in response to a system interrupt, or just in response to certain system interrupts. In a similar manner, in the case of exceptions, like invalid memory access, protection violation, integer overflow, or division by zero, the mechanism of the present invention could be easily adapted to raise performance levels to assist in more quickly responding to exceptions.

Based on the mechanism of the present invention, some consideration should be given to the effect on processor performance that will result from traditional manipulations of task
priority. Again, task priority is not a fixed attribute. Most systems offer a service that changes task priority. Attention should be paid to changing the performance value along with changes to task priority, unless circumstances exist that would not warrant this corresponding change. In other circumstance attention may be required to the impact on priority inversion avoidance schemes. Mechanisms like priority inheritance protocol and priority ceiling protocol change a task’s priority depending on the state of any locks that tasks may be holding. For instance, a task with a high priority may have to wait for a task with a lower priority to finish because the lower priority task has a lock on a system resource needed by the higher priority task. Systems use various protocols to temporarily boost the priority of the lower priority task to minimize the wait time of the higher priority task. With the mechanism of the present invention these protocols would need to also determine if processor performance should change along with task priority. Another aspect of changing priority involves systems that implement aging mechanisms by which tasks that are ready to run are assigned a scheduling priority that gradually increases over time so that even low-priority tasks eventually get some processor time. In this case, the task will be scheduled according to the aged priority, but will run at its original lower priority. With the mechanism of the present invention, the system would need to choose between letting the performance level increase according to the increasing task priority, or locking the performance level while allowing the task priority to temporarily elevate based on age. Finally, some consideration should be given to the impact of the present invention on scheduling algorithms. Since the processor performance level is no longer fixed, task completion time can vary. Scheduling algorithms that use estimates of a task’s run time will need to account for the fact that run time will vary with task priority as that value will now impact processor performance level.
For example, rate-monotonic analysis (RMA) for real-time scheduling sets task priority in proportion to the frequency at which the task occurs. Since this invention makes higher priority tasks run faster the fundamental principles of RMA will continue to hold. Provided that the performance does not decrease with increasing priority RMA continues to work, however, advanced forms of RMA that rely on estimates of a task’s runtime will need to take into account the impact of a task’s priority level on run time.

As indicated above, one of the primary motivations for varying processor performance relates to managing the performance versus power consumption trade-off, and the corresponding performance versus heat dissipation trade-off. While the present invention is applicable to the management of this problem, it is not so limited. The invention is applicable without significant alteration to manage any form of performance trade-off that is operative at run time, like, performance versus system lifetime, performance versus emission level, and the like.

The present invention is particularly well suited for application to embedded real-time operations, although it is not limited thereto. In embedded systems, there is a premium on efficient use of system resources. Embedded computer systems usually operate in environments that limit the available amount of memory, processor power, and limit the size of the systems used. Furthermore, real-time systems must often perform a task in a specified period of time. The wide area of emerging consumer electronics products and Internet appliances often operate remotely and under battery power. These constraints place enormous demands on real-time systems and create a need to pay special attention to efficiency of operation without sacrificing reliability. Embedded systems do not have the luxury of solving these problems by simple adding more power or more memory, in the style
of personal computers. Thus, the present invention offers a method to use task priority to scale processor performance particularly useful in embedded real-time applications.

The foregoing description and drawings comprise illustrative embodiments of the present inventions. The foregoing embodiments and the methods described herein may vary based on the ability, experience, and preference of those skilled in the art. Merely listing the steps of the method in a certain order does not constitute any limitation on the order of the steps of the method. The foregoing description and drawings merely explain and illustrate the invention, and the invention is not limited thereto, except insofar as the claims are so limited.

Those skilled in the art that have the disclosure before them will be able to make modifications and variations therein without departing from the scope of the invention.
Claiming:

1. A method of using task priority to scale processor performance, said method comprising:
   providing a general-purpose computer for performing tasks with a processor capable of operating at a plurality of performance levels, and has the capability to set a plurality of task priority levels;
   reading a task’s priority level;
   associating said task’s priority level with a performance level; and
   setting said processor to operate at said performance level.

2. The invention in accordance with claim 1 further comprising the step of creating a priority map whereby said operating system associates said plurality of task priorities with said plurality of performance levels, and wherein said step of associating said task’s priority level with a performance level and said step of setting said processor to operate at said performance level are performed using said map.

3. The invention in accordance with claim 2 wherein said general-purpose computer further comprises a performance control driver to facilitate communication between said operating system and hardware components of said general-purpose computer, and further comprising the step of calling said performance control driver to convert said map in to a performance value specific to said hardware components of said general-purpose computer, and wherein said step of setting said processor to operate at said performance level is accomplished by calling said performance control driver.
4. The invention in accordance with claim 1 wherein the step of associating said task’s priority level with a performance level is accomplished by updating said task’s task descriptor with said performance level.

5. The invention in accordance with claim 1 wherein changing a clock speed sets said performance level of said processor.

6. The invention in accordance with claim 1 wherein changing said processors voltage set said performance level of said processor.

7. The invention in accordance with claim 1 wherein said computer has an embedded real-time operating system.
FIGURE 1

Return an Error

Is a mapping specified?

Yes

Return an Error

Is the mapping valid?

Yes

Convert the specified mapping into a map of priority to unsigned integer performance values.

Call the performance driver to convert the map of unsigned integer performance values to system specific performance values specific to the driver.

Store the map for later reference.

Return success
FIGURE 2

New priority == old priority?

Find performance_val pointer for current priority in the performance map.

Store the performance_val pointer in the target task's descriptor.

Target_task == current task?

yes

Force context switch processing to cause call to performance modification

Done
FIGURE 3

Get the task performance field from the task's descriptor

Call the performance driver passing it the performance field from the previous step

Is error logging available?

No

Log the error

No

Returned success?

No

Continue with context switch
FIGURE 4

New_value == saved_value

New_value valid?

Adjust hardware to the performance level set in new_value.

Hardware manipulation successful?

Saved_value = new_value

Return success

No

Return error
FIGURE 5

For each value in in_table

Entry valid?

No

Return error

Allocate memory for performance_val structures

For each value in in_table

Convert in_table[I] to hardware-dependent performance_val in out_table[I]

Return success
## INTERNATIONAL SEARCH REPORT

### A. CLASSIFICATION OF SUBJECT MATTER

<table>
<thead>
<tr>
<th>IPC(7)</th>
<th>G06F 09/00</th>
</tr>
</thead>
<tbody>
<tr>
<td>US CL</td>
<td>709/103</td>
</tr>
</tbody>
</table>

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)

U.S.: 709/103, 107

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 Practicable, search terms used)

Please See Continuation Sheet

### 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>Y</td>
<td>US 5,623,647 A (MAITRA) 22 APRIL 1997 (22.04.1997) Fig. 3-7</td>
<td>1-7</td>
</tr>
<tr>
<td>P,Y</td>
<td>US 6,272,544 B1 (MULLEN) 07 August 2001 (07.08.2001) col. 1 line 55 - col. 3, line 65</td>
<td>1-7</td>
</tr>
<tr>
<td>Y</td>
<td>US 6,092,095 A (MAYTAL) 18 July 2000 (18.07.2000) col. 2-4</td>
<td>1-7</td>
</tr>
</tbody>
</table>

☐ Further documents are listed in the continuation of Box C.  ☐ See patent family annex.

<table>
<thead>
<tr>
<th>Special categories of cited documents:</th>
<th>&quot;T&quot;</th>
<th>&quot;X&quot;</th>
<th>&quot;Y&quot;</th>
<th>&quot;&amp;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>document defining the general state of the art which is not considered to be of particular relevance</td>
<td>&quot;T&quot;</td>
<td>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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>earlier application or patent published on or after the international filing date document which may throw doubts on priority claims or which is cited to establish the publication date of another citation or other special reason (as specified)</td>
<td>&quot;X&quot;</td>
<td>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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>document referring to an oral disclosure, use, exhibition or other means</td>
<td>&quot;Y&quot;</td>
<td>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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>document published prior to the international filing date but later than the priority date claimed</td>
<td>&quot;&amp;&quot;</td>
<td>document member of the same patent family</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Date of the actual completion of the international search: 05 July 2002 (05.07.2002)

Date of mailing of the international search report: 09 AUG 2002

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703)305-3230

Authorized officer
Meng-Ai M An

Telephone No. 703-305-3900

Form PCT/ISA/210 (second sheet) (July 1998)
Continuation of B. FIELDS SEARCHED Item 3:
EAST, IEEE Online
search terms: Priority level, performance level, scheduling, dynamic, progress