DPM at OS level: low-power scheduling policies

C.BRANDOLESE, W.FORNACIARI, F.SALICE
Dipartimento di Elettronica e Informazione
Politecnico di Milano
P.zza L.Da Vinci, 32 – 20133 Milano, ITALY

L.POMANTE
Univ. degli Studi de L'Aquila-DEWS,
67040 Poggio di Roio
(AQ), ITALY

R.ZAFALON
STMicroeletronics
AST - Research &
Innovation
Via C.Olivetti, 2
20041 Agrate Brianza,
Milano, ITALY

Abstract: - Reducing power consumption has become one of the major goals in designing electronic systems for mobile devices and embedded applications. One of the most promising approaches, called Dynamic Power Management (DPM), is to adapt at run-time power states of system components by means of changing the operating voltage of a processor as well as switching I/O devices to low-power sleeping states during periods of inactivity. This work is aimed to provide innovative strategies to enhance the capabilities of a potential Power Manager that acts at the operating system level. In particular, several low-power process scheduling policies, increasing the possibility to conveniently force devices into low power states, have been implemented in the Linux operating system to verify their feasibility and to analyze their benefits and drawbacks.

Key-words: Power Optimization, System Modeling, Operating Systems, Scheduling, Device Optimization

1 Introduction

Power management is becoming a central issue for embedded systems. Power demand and the corresponding energy consumption, of the various components of an embedded system, directly influences battery lifetime, hence the systems performance/lifetime. As such systems are deployed in different operating conditions they should be able to self-adapt by controlling power vs. performance trade-off [1].

Electronic systems can be viewed as collections of components, each consuming a fraction of the power budget, which are not always required to be in the active state under a purely functional standpoint [2][4].

Techniques oriented to enable and disable components, as well as adapting their performance to the workload, are called *Dynamic Power Management* (DPM) techniques. These ones can make use of several features supported by modern hardware designs, including multiple power states in I/O devices and variable-voltage processors [3][13][14], and can be implemented at different abstraction levels. Recently, the targeting of DPM strategies towards the operating system (OS) level has gained importance due to its flexibility and ease of use [18]. In fact, because of the OS has an overall view of the system resources and workload trend, it is possible to take customized power-management decisions and, as a consequence, achieving significant energy savings.

In this context, a more general project [16][17] aims to develop innovative techniques at the OS level to allow Dynamic Power Management of Input/Output devices. Within the framework of such a project, this paper focuses on the implementation and experimental validation of innovative *low-power* scheduling policies for power management at the operating system level. In particular, the paper proposes different process-level scheduling policies, providing valid opportunities to a *Power Manager* for switching devices to low power states with higher probability to obtain a real advantage. The different scheduling policies have been implemented in the Linux operating system and tested on an x86 platform, analyzing the scheduling functionalities and the provided power saving opportunities.

This paper is organized as follows: Section 2 analyzes the main DPM approaches reported in literature, Section 3 illustrates the proposed innovative scheduling policies and Section 4 describes the test results for each of the implemented techniques. Finally, Section 5 draws some conclusions and depicts the future goals of the overall project.

2 Related Work

Low power design has been addressed in recent researches at various levels of abstraction ranging from system level to physical level. DPM techniques reported in literature can be classified in three main levels: *application-operating system interface*, *operating system* and *operating system-hardware interface*.

At the *application-operating system level*, it is possible to apply power optimization by integrating the application layer into dynamic power management of devices. These techniques exploit I/O devices to save energy. For example, in [9] a new OS interface is introduced for cooperative I/O that can be exploited by energy-aware applications, while in [8] this is achieved by introducing new system calls which allow interactive applications to inform the OS about future device requests: this enable a proper schedule of the processes and, consequently, a power reduction

At *operating system level* it is possible to search for tasks requiring services from hardware components, as sources of power consumptions. OS has detailed knowledge about running tasks, so that this information can be used for power management. A DPM scheme at OS level that adapts power state of hardware components depending on workload, can deal with problems that in other levels cannot be handled [5] and can identify time intervals where I/O devices are not being used and switch these devices to low power state [3][14].

A significant effort is spent toward techniques that act at the *operating system-hardware level*, namely, that try to apply, at the same time, OS techniques based on particular hardware architectures. In this wide class we can distinguish between the specialized hardware that can be typically a microprocessor or a memory. In literature there are several techniques, namely scheduling algorithms, for variable voltage selection microprocessors [6][10][11] or page replacement algorithms for *Rambus off-chip* memory [7].

The work presented in this paper focuses exclusively on the operating system level in order to overcome the obstacles of platform portability, application transparency and independence from hardware features.

3 Low Power Scheduling

I/O-centric DPM techniques at the operating system level take advantage from the information regarding running tasks. In particular, depending on process device utilization, the *Power Manager* can decide which devices has to move in the low power states available at run-time. Obviously, varying the device power states imply considering possible energy overhead, mainly related to *break-even time* (i.e., the min length of idle time to achieve power saving [8][17]). It is crucial to find out a trade-off between the energy saved maintaining the devices in the low power states and the energy spent during state transition. For this reason, a valuable aid can be provided by the interaction with the process scheduler, which could handle tasks execution in such a way to enlarge idle periods for existing I/O devices.

Based on such a background, this section aims at describing the proposed innovative *low-power* scheduling policies able to increase power saving opportunities for a *Power Manager*. We partitioned the algorithms in two different classes: *device-oriented scheduler* and *device-exclusion scheduler*. This choice has been suggested by the differences in the principles on which the two approaches are based on.

3.1 Device-Oriented Scheduler

The device-oriented scheduler aims to select processes depending on their devices utilization. In particular, the policy tries to organize the execution of processes to obtain a certain clustering in the device utilization [8].

Let us assume that three processes are in the ready state on a certain system. These three processes, pc_1, pc_2, pc_3, use devices from a set composed of d_1 and d_2. A classic priority-based scheduling algorithm could organize process execution as depicted in Figure 1.

In such a case, the time intervals in which a device is idle appears to be short and with a very fragmented distribution over the time. In such a way, there will be few opportunities to save power by mean of forcing the devices in low power states.

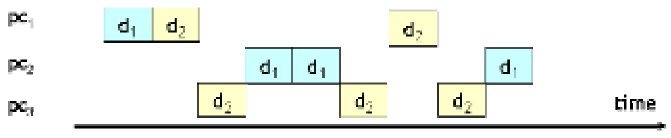

Figure 1 - Priority-based scheduling

On the contrary, if the scheduler selects the next process to be executed considering the devices required, the device utilization diagram could present longer idle time intervals, hence resulting in more opportunities to save power, as shown in Figure 2.

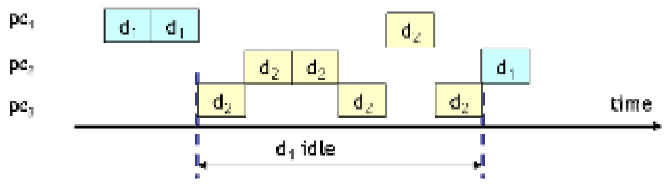

Figure 2 – Device-oriented scheduling

As already recalled, the device-oriented scheduler considers the processes in the ready queue and selects the next to be executed depending on the device utilization. Clearly, different policies could influence the scheduling behavior in such a way that each one provides different results for the device utilization. To this purpose, different device-oriented policies are detailed in the following.

3.1.1 Maximum Common Devices

The *Maximum Common Devices* (MCD) policy suggests to select the next process to be executed on the basis of the number of devices required which are in common with the current process. Consequently, long time intervals, during which processes using a particular set of devices are executed, correspond to long time intervals in which other devices are idle. This idea is summarized in the algorithm reported in Figure 3.

Error! Objects cannot be created from editing field codes.

Figure 3 – MCD algorithm

This algorithm has been implemented in the Linux kernel. The device set used by each process is retrieved by looking at a particular data structure, the *process descriptor*, which is used by the kernel for managing processes. Consequently, the scheduler looks at this data structure to determine which kind of devices each process use.

It is worth noting that several checks have been added to the basic algorithm in order to avoid deadlock and starvation.

3.1.2 Minimum Not Common Devices

A dual policy, named *Minimum Not Common Devices* (MNCD), has been used to implement a different device oriented scheduler in the kernel. While MCD selects the next process which uses the maximum number of devices in common with the current process, MNCD scheduler selects the next process to be executed on the basis of the number of devices required, which are newer than the required device set of the current process. Consequently, the MNCD scheduler, because it looks for the next process which uses the smallest set of devices not in common with the current process, maximizes the time intervals during which the devices remain idle. The related algorithm is described in Figure 4 while the implementation issues are analogous to the MCD ones.

```
for every ready process do
  if task is not the current
    for all devices used
      if device is not used by the current process
        increase a counter;

select the process with the minimum counter value;
if it is possible schedule it;
else schedule normal;
```

Figure 4 – MNCD algorithm

3.1.3 MCD and MNCD Integration

Both MCD and MNCD policies try to maximize one aspect: time intervals where devices are busy or time intervals where devices are idle, respectively. A meaningful improvement can be the integration these two approaches. It is actually possible to think that the scheduler looks for the next process using the minimum new device set, and that when it identifies such processes, it selects the one with the maximum number of devices common with the current process. The resulting algorithm is shown in Figure 5.

```
for every ready process
  if task is not the current
    for all the device used
      if device is used by current process
        increase the common device counter;
      else if device is not used by current process
        increase the new device counter;

select if possible the process with min new devices
counter and the max common device counter;
```

Figure 5 – MNCD-MCD algorithm

3.2 Device-Exclusion Scheduler

With the device-oriented schedulers, the scheduler selects next processes on the basis of their device requirements. This approach has the aim to adapt the device utilization dynamically, in order to create power saving opportunities. An opposite approach is to determine a fixed pattern of utilization for a particular device set. The pattern has to be known by the scheduler that will select next processes depending on which device can be used and which not, as specified by the device utilization pattern. Therefore, the device exclusion oriented approach knows *a priori* (i.e. the *Power Manager* could communicate such a decision to the scheduler) which devices to switch to the low power state and for how long. In this way, the scheduler will select a process to run on the basis of which devices are allowed to be in the busy state.

Such an *extreme approach* arises for the consideration that using a device-oriented scheduler, the device idle times are not deterministically known and cannot be controlled. Instead, with a device-exclusion scheduler, the idle intervals for a device could be set to be greater than its *break-even time* (i.e., the minimal length of idle time to achieve power saving [8][17]) so avoiding wasting energy because of transition energy overhead.

In fact, it is possible to think that the scheduling algorithm takes a decision for the next process which does not use a particular device given a device utilization sequence, which is known by the algorithm. This approach has been called *device exclusion scheduler*. It should be said that this technique could be inefficient in heavily interactive systems in which the workload is unknown until run time. The device exclusion oriented algorithm is shown in Figure 6.

```
if timeout value has expired
  change device excluded

for every ready process
  if task is not the current
    for all the device used
      if device excluded is used
        does not schedule it;
      else schedule normal;
```

Figure 6 – Device-exclusion algorithm

For the sake of clearness, the algorithm implemented in this work does consider only a fixed set of device and a sequence with constant time intervals. In particular, the time length during which a device is not used has been set to *1 s*. That is, the scheduler knows that each second a particular device cannot be used and then select next processes consequently. To achieve this goal, the algorithm uses the `jiffies` variable, which has the value of the absolute machine time since the last start.

4 Experimental Results

The policies and the algorithms discussed above, have been implemented in the Linux OS (kernel 2.4.25) and tested on a desktop computer with an *Intel Pentium II* processor. Test suites have been created selecting programs from a benchmark suite freely available, named *MiBench*. We have chosen to use programs from the *Consumer Devices category*, because they are focused primarily on multimedia applications. In the following, the obtained results are discussed both qualitatively and quantitatively and compared to the unmodified classical priority-based operating system scheduling strategy.

4.1 Device-oriented scheduler

The first tests have been done by considering the MCD scheduling algorithm. The devices investigated have been the *sound card* and four partitions of the secondary hard disk. It is worth noting that the four partitions behave as they were distinct devices.

As is it possible to see from Figure 7 and Figure 8 the new scheduler produces an increasing of both the maximum and the mean idle time lengths.

Figure 9 and Figure 10 depicts the maximum and mean idle time diagrams obtained with the MNCD scheduler.

It is clear that the idle time values enhancements obtained are better than the values provided by the MCD scheduling algorithm. Consequently, it is possible to state that the MNCD scheduler has a better capability in creating power saving opportunities. An explanation is the fact that MNCD scheduler directly acts on the devices being idle trying to extend time intervals during which devices remain idle, whereas MCD algorithm has the aim to extend time intervals during which devices are busy.

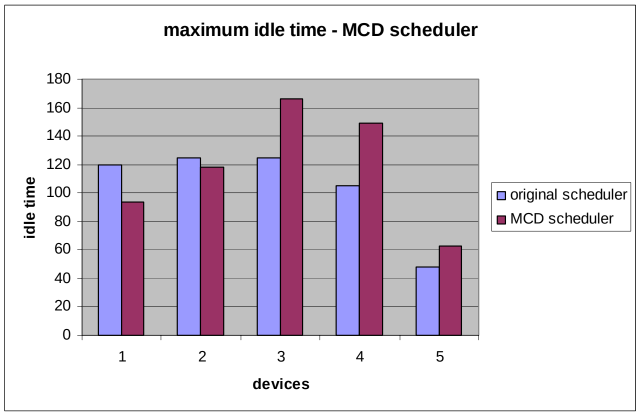

Figure 7 - Max idle time (ms) – MCD algorithm

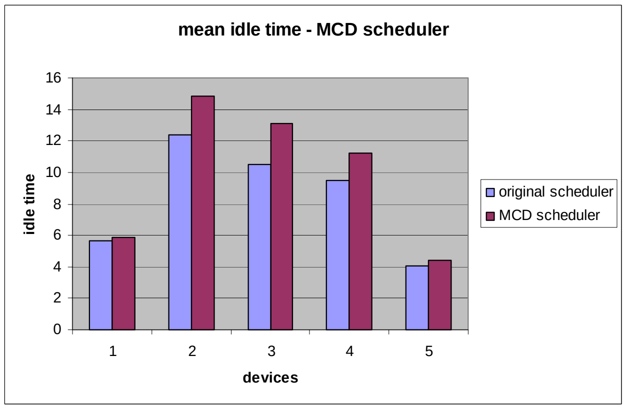

Figure 8 - Mean idle time (ms) – MCD algorithm

The last test campaign we have considered has been the integration of the two above discussed scheduling methods. The results have suggested considering a new scheduling algorithm that first checks the number of not common devices, and then the number of common devices with the current process. The results found are shown in Figure 11 and Figure 12. As we expected, the idle times are longer, for both mean and minimum values.

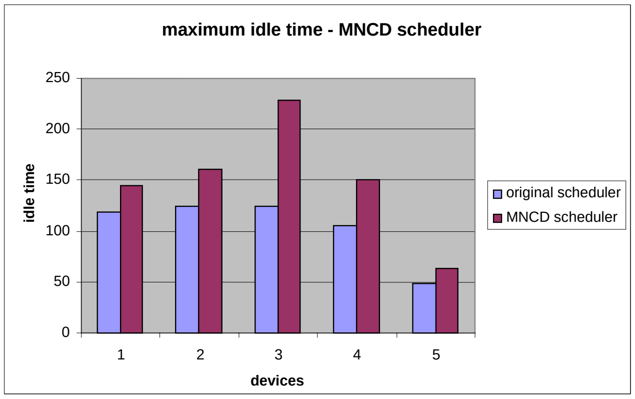

Figure 9 - Max idle time (ms) – MNCD algorithm

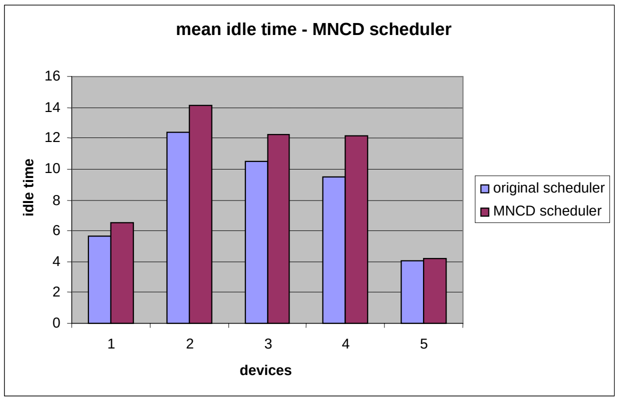

Figure 10 - Mean idle time (ms) – MNCD algorithm

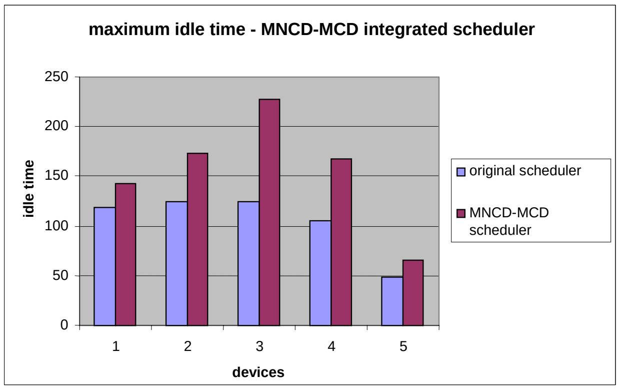

Figure 11 - Max idle time (ms) – MNCD+MCD

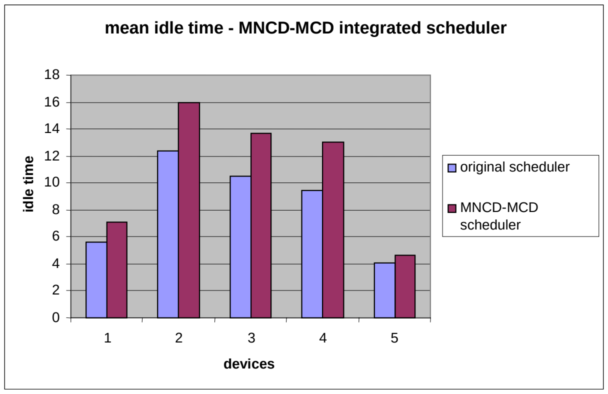

Figure 12 - Mean idle time (ms) – MNCD+MCD

4.2 Device-exclusion scheduler

The device exclusion scheduling algorithm has also been implemented in Linux and tested with the general purpose platform. We have considered similar test and data analysis procedures of what previously adopted for the device oriented algorithms.

To test the correct functionality of the modified scheduling algorithm, a shut-down time interval of 1 second has been chosen. Such an exclusion rule has been applied to a particular set of devices, constituted by physical devices such as the *Open Sound System Digital audio* and the *PS/2-style mouse*, and by logical devices such as the *TTY devices first virtual console* and the *Alternate TTY device System Console*.

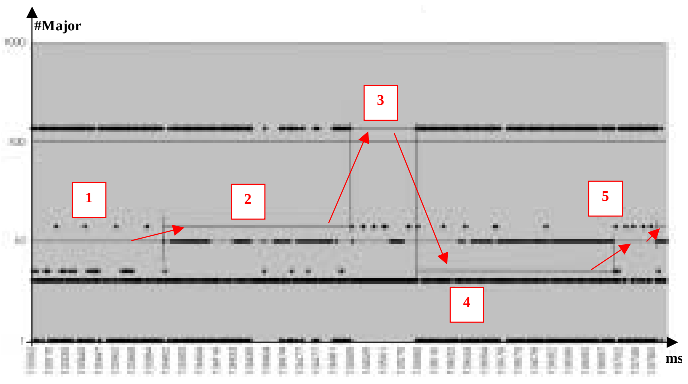

Figure 13 – Device utilization diagram

Figure 13 shows the distribution over the time of the device utilization by scheduled processes. It is worth noting how devices are not used periodically in correspondence to the shut down time intervals (it must be noted that the time axis is not linearly scaled): in the first interval (1) the device with #Major=10 is not used by any scheduled process, then (2) it is the time of the one with #Major=14 and so on to start another cycle (5) excluding again the device with #Major=10. Unfortunately, such an approach, used stand-alone, could limit the reactivity of the system that become difficult to use.

5 Conclusions

This work proposed innovative strategies to provide opportunities for dynamic power saving in embedded systems. Several scheduling techniques, which provide power saving opportunities for a potential *Power Manager* has been proposed. In particular, two approaches have been identified: *device-oriented scheduling* and *device-exclusion scheduling*. Tests for all the policies considered have shown that the most promising results for power saving are provided by the algorithm which integrates the MCD and the MNCD techniques. The device-exclusion scheduling algorithm implemented revealed that such an approach is not suitable, if used stand-alone, for reactive systems. Current effort is focusing on the definition of a *Power Manager* module able to take advantage of the power saving opportunities provided by the implemented scheduling algorithms really forcing the physical devices in low-power states. Moreover, new scheduling algorithms could be integrated in the framework, which allows full integration with classic priority-based and with device-exclusion oriented approaches.

References

[1] A. Bogliolo, L. Benini, E. Lattanzi, G. De Micheli, Specification and Analysis of Power-Managed Systems, *Proceedings of the IEEE, Vol. 92, No. 8*, August, 2004.

[2] Flavius Gruian, Microprocessors: Low Power and Low Energy Solutions, *course paper for Advanced Issues in Computer Architecture*, Spring, 1999.

[3] L.Benini, A. Bogliolo, G. De Micheli, A Survey of Design techniques for System-Level Dynamic Power Management, *IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 3,* June, 2000

[4] K. Masselos, F. Catthoor, C. E. Goutis, H. DeMan, Low Power Mapping of Video Processing Applications on VLIW Multimedia Processors, *IEEE Alessandro Volta Memorial Int. Workshop. on Low Power Design (VOLTA)*, Como, Italy, pp.52-60, March, 1999.

[5] Y.H. Lu, L. Benini, G. De Micheli, Operating System Directed Power Reduction, in *Proc. Int. Symp. Low Power Electronics Design*, Rapallo, Italy, July, 2000.

[6] IBM and MontaVista Software, Dynamic Power Management for Embedded Systems, November, 2002

[7] R. Lebeck, Xiabo Fan, Heng Zen, C. Ellis, Power Aware Page Allocation, In *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2000.

[8] Y.H. Lu, L. Benini, G. De Micheli, Power-Aware Operating Systems for Interactive Systems, *IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 10, No. 2,* April, 2002.

[9] A. Weissel, B. Beutel, F. Bellosa, Cooperative I/O – A novel I/O semantics for Energy-Aware Applications, *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December, 2002.

[10] Yumin Zhang, Danny Z. Chen, Task scheduling and voltage selection for energy minimization, *annual ACM IEEE Design Automation Conference Proceedings of the 39th conference on Design automation*, 2002.

[11] G. Quan and X. Hu, Energy efficiency fixed priority scheduling for real-time systems on variable voltage processors, *ACM/IEEE Design Automation Conference*, June, 2001.

[12] L.C. Weng, X. Wang, B. Liu, A survey of Dynamic Power Optimization Techniques, *Proceedings of The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*, 2003.

[13] Y.-H. Lu, and G. D. Micheli, Comparing System-Level Power Management Policies, *IEEE Design & Test of Computers*, 2001.

[14] R.Golding, P.Bosh, and J.Wilkes, Idleness is not sloth, In *Proc. USENIX Winter Conf.*, New Orleans, 1995.

[15] Daniel P. Bovet and Marco Cesati, Understanding the Linux Kernel, O'Reilly, First edition, October 2000.

[16] R. Merlani, Run Time Power Management based on Real Time Operating System, Final Report, XVI Master IT, CEFRIEL, Italy, 2004.

[17] V. Di Maria, Run Time Power Management at Operating System Level, Final Report, XVII Master IT, CEFRIEL, Italy, 2005.

[18] R. Zafalon, Paolo Bacchetta, RT-OS Run Time Power Management for Mobile Terminals, Embedded Systems Conference, March 2005, San Francisco, USA.