17th IEEE Real-Time Systems Symposium: Work in Progress Sessions

Archer, Myla
Boston University Computer Science Department

http://hdl.handle.net/2144/3747
Boston University
The Design and Implementation of the CPU Power Regulator for Multimedia Operating Systems

Giun-Haur Huang, Shie-Kai Ni, and Tei-Wei Kuo
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan 621, ROC

Abstract

This paper describes a Windows NT/95 utility, the CPU Power Regulator (CPR), which improves the capability of Windows NT/95 in servicing time-critical applications. CPR considers a distance model [4] to service time-critical applications such as multimedia softwares and electronic games in a timely fashion. Distinct from the past work [7, 8, 9], CPR adopts a user-level control mechanism to manage the resource allocations on Windows NT/95 and makes no modifications to the operating system and application softwares. The performance of CPR was verified by a collection of simulation experiments of randomly generated and realistic workloads. CPR not only introduces very low system overheads but also largely reduces the phenomenon of non-timely resource allocation for applications. The experimental results also demonstrate the capability and flexibility of CPR in multiplexing CPU cycles to provide different degrees of quality-of-service to time-critical applications. The results of this work present a low-cost software solution to transform an ordinary operating system into a multimedia operating system.

1 Introduction

With the advent of new technologies such as multimedia and high-speed network, there is increasing demand for computer systems with stringent timing constraints. Among the many types of applications for which time-constrained and predictable response is required, the most familiar are multimedia applications and electronic games. These time-critical applications depend on the operating system to allocate resources in a timely fashion. Although these time-critical applications are used so frequently in our daily life, many commercial operating systems such as Microsoft Windows NT and 95 still do not possess the capability to allocate resources in a timely fashion [5, 10, 11].

This paper describes a Microsoft Windows NT/95 utility named CPU Power Regulator (CPR) which allows an application to reserve an amount of CPU cycles for its run-time usages. CPR considers a distance model [4], in which each application \( \tau_i \) is serviced at a fixed interval \( D_i \), as shown in Figure 1. CPR is responsible to properly multiplex CPU cycles among applications in a periodic way. For implementation, CPR adopts a user-level control mechanism based on the idea of time reservation [7, 8, 9] to multiplex CPU cycles among applications. Distinct from the past work, CPR makes no modifications to the operating system or any application program. CPR considers a model of resource reservation at the application level, which is more natural for ordinary users and better for modular design of complex systems. The performance of CPR was verified by a collection of simulation experiments of randomly generated and realistic workloads. CPR not only largely reduces the phenomenon of non-timely resource allocation for applications but also consumes less than 5% of the total CPU cycles, in the worst case on a Pentium 90MHZ personal computer. The experimental results also demonstrate the capability and flexibility of CPR in multiplexing CPU cycles to provide different degrees of quality-of-service to time-critical applications. CPR is a part of the video-on-demand (VOD) project at the Advanced System Integration Laboratory at the National Chung Cheng University. Its main responsibility is to guarantee the timeliness of the video servers in pumping each individual MPEG stream into the ATM network and the

---

*Supported in part by research grants from the National Science Council under Grant NSC85-2213-E-194-014 and NSC86-2213-E-194-007

To the best of our knowledge, the best CPU monitor utility such as perfmon.exe already takes 3% of the total CPU time, in the worst case on a Pentium 90MHZ personal computer. Note that the sampling rate is set at 10ms.
response time of the connection management of the
servers.

2 System Overview

2.1 System Architecture

![Figure 1: Application \( \tau \) under the distance model](image)

We define that an application is a collection of user/system threads that work for a common goal, and an application is the smallest unit of CPU reservation in CPR. CPR allows applications to reserve CPU cycles in a periodic way, as shown in Figure 1. That is, the threads of an application may totally execute for \( C \) time units of CPU cycles at a fixed interval of \( D \) time units. Note that the scheduling of threads in the same application is not the concern of CPR. CPR will retain or try to retain the execution order of the threads in their original order without CPR. This is mainly accomplished by not altering the priority hierarchy of the threads (when CPR does not exist).

![Figure 2: The system architecture of CPU Power Regulator](image)

As shown in Figure 2, CPR consists of five modules: user interface, admission control, policy enforcement, feedback control, and resource monitor. A user

may make a CPU reservation for a time-critical application through a user-friendly interface. Once the reservation request passes the admission control, the policy enforcement module and the feedback control module will provide the application with the reserved CPU cycles in a timely fashion. The responsibility of the resource monitor is to track the CPU usage of an application to prevent it from overrunning or starvation.

For the implementation, CPR adopts a user-level control mechanism based on the idea of time reservation [7, 8, 9] to multiplex CPU cycles among applications. Prior to further discussions of this work, we should point out the major difficulties which CPR faced at the design and implementation phases, comparing with the past work [7, 8, 9]. Note that CPR has no access to the source code and most of the internal system information of Windows NT.

- The time granularity accessible by programmers is large, and the alarm facilities, especially when low system overheads are required, are not timely and reliable.
- Most system information maintained in Windows NT is protected and not accessible by users. For example, the threads, especially the user threads and system threads, of an application may not have any parent-child or sibling relation, and their relationship is encapsulated in the system without any access from ordinary users at all.
- Many resource scheduling policies on commercial operating systems including Windows NT are not suitable to time-critical applications. For example, Windows NT solves the priority inversion problem by randomly boosting threads’ priorities.
- Application users rarely know how many or which threads constitute an application. Not to mention, ordinary users even don’t know how much a system resource such as CPU cycles should be reserved for each thread. Even if users know the resource requirements of application, they usually have no rights to modify the source codes of application programs.

These difficulties not only make the resource monitoring and controlling of threads very challenging but also let the system models and resource management mechanisms proposed by the past work not suitable to our needs. A user-friendly and flexible interface for CPR is also a necessity for its success on Windows NT. In particular, we need a system model more suitable to
our application domain and an adaptive and flexible control mechanism. Most importantly, the overheads of CPR must be low and controlled, and its performance must be justifiable.

2.2 Policy Enforcement and Resource Monitor

The time reservation \((C_i, P_i)\) of each application is considered as an independent periodic process scheduled by CPR. CPR uses the Rate Monotonic Scheduling (RMS) policy [6] to schedule the reservations of applications mainly for two reasons:

1. RMS suffers less jitter problems than many scheduling algorithms such as the Earliest Deadline First scheduling algorithm (EDF) [3, 6], and RMS, therefore, better satisfies the jitter requirements of the distance model.

2. A fixed-priority assignment scheme such as the one adopted by RMS is better for CPR to maintain the original priority hierarchy of the threads in an application.

Because CPR schedules the CPU reservation \((C_i, P_i)\) of each application as an independent periodic process, the policy enforcement module of CPR must try to meet the “timing constraints” \((C_i, P_i)\) of each application. The policy enforcement module uses RMS to schedule the time reservations \((C_i, P_i)\) of applications. Whenever a time reservation with a higher priority (i.e., a smaller \(P_i\)) than the current “running” time reservation is “ready”, the policy enforcement module will raise the class of the corresponding application to “High” and lower the class of the current “running” application to “Idle”, where the classes of the other applications remain the same. Note that there is only one application with a priority class equal to “High” at any time, all of the other applications have a priority class equal to “Idle”. When the class of an application is changed, the priorities of the user threads of the application are changed too, because the priority level of a thread is a function of the base priority of its process (application)\(^3\).

Resource monitoring of each individual application is an extremely difficult job on Microsoft Windows NT because the threads, especially the user threads and system threads, of an application may not have any parent-child or sibling relation, and their relationship is encapsulated in the system without any access from ordinary users at all. The resource monitor of CPR tackled the thread relationship (belonging) problem by tracking the creation of new threads since Microsoft Windows NT usually forks new threads to service applications\(^4\). The tracking of thread creations is done mainly by injecting our Dynamic Link Library (DLL) into Windows NT. Whenever a thread is forked, the injected DLL will trigger a corresponding routine in the resource monitor to determine the relationship of the newly forked thread with the currently executing applications. The DLL injection method is also supplemented with a quick scanning process over Windows NT’s registry database, especially when applications are being initialized, for more precise identification of new threads and thread relations.

2.3 Performance Evaluation

The performance of CPR was verified by a collection of simulation experiments of randomly generated and realistic workloads. CPR not only consumes no more than 5% total CPU cycles of a Pentium 90 MZH personal computer but also largely reduces the phenomenon of non-timely resource allocation for applications. With limited space, we only present some of the simulation results from [3],

\[\begin{array}{ccccccccc}
\text{Frames Per Second} & & & & & & & & \\
\text{NT Opt. 1} & \text{NT Opt. 2} & \text{NT Opt. 3} & \text{CPR} & \\
\hline
0 & 5 & 10 & 15 & 20 & 25 & 30 & \\
\end{array}\]

Figure 3: The frames per second of ANIMAL1.MPG

Figure 3 shows that the foreground VMPEG Ver-

\(^3\)The real-time scheduling mechanism adopted by CPR is similar to the mechanism proposed by Adelberg, et al. [1], whereas CPR adopts a different priority structure of applications and allows virtually infinite relative priority levels among applications. In particular, CPR raises or lowers the priorities of multiple threads of an application at a time.

\(^4\)Microsoft Windows NT depends on a Client-Server architecture to service applications [2]. Whenever system services are needed by applications, new system threads are usually forked. However, certain system threads such as those which dispatch keyboard and mouse events exist to provide general services to applications, but they usually consume very limited CPU cycles.
sion 1.7 played movies with a quality almost as bad as the background VMPEG Version 1.7 did on Windows NT without CPR. The foreground and background movies suffered about the same degree of frame skippings, no matter which scheduling option of Windows NT was adopted\(^5\). However, CPR consistently guaranteed the service quality of the chosen (foreground) VMPEG Version 1.7 with a time reservation \((40\, \text{ms}, 50\, \text{ms})\). Note that a time reservation with an even higher \(C_i/P_i\) ratio may further push the Average Frames Per Second of each movie playing to approach the maximum Average Frames Per Second of the movie possibly on the test personal computer. (The maximum Average Frames Per Second of ANIMAL1.MPG played by VMPEG Version 1.7 is 24.01.)

3 Summary

This paper describes a Windows NT/95 utility, the CPU Power Regulator (CPR), which improves the capability of Windows NT/95 in servicing time-critical applications. CPR adopts a user-level control mechanism to manage the resource allocations on Windows NT/95 and makes no modifications to the operating system and application programs. CPR tackles various design and implementation difficulties on Windows NT/95 such as unreliable timers and inaccessibility of system information. We propose a larger granularity of resource reservation at the application level, as well as flexible and feedback control mechanisms suitable to applications running on commercial operating systems. The performance of CPR was verified by a collection of simulation experiments of randomly generated and realistic workloads. CPR not only consumes no more than 5% total CPU cycles of a Pentium 90MHZ personal computer but also largely reduces the phenomenon of non-timely resource allocation for applications. The experimental results also demonstrate the capability and flexibility of CPR in multiplexing CPU cycles to provide different degrees of quality-of-service to time-critical applications [3].

This paper presents a low-cost software solution in building a multimedia environment on commercial operating systems, and the solution will be more and more attractive as CPU speed increases rapidly. For future research, we will further extend the capability of CPR to control system resources other than CPU cycles and to manage time reservations more precisely. We also want to extend the results of this work to multiprocessor environments and further develop CPR into a full prototyping, synthesis, and simulation environment.

References