Design and Implementation of Accounting System for Information Appliances

Midori Sugaya [1], Shuichi Oikawa [2], Tatsuo Nakajima [1]

[1] Department of Computer Science, Waseda University
[2] Department of Computer Science, University of Tsukuba
{doly,tatsuo}@dcl.info.waseda.ac.jp, shui@cs.tsukuba.ac.jp

Abstract. This paper presents the design and implementation of Accounting System that is a resource monitoring and restriction system. The system improves the reliability and security of a system. Accounting System is a generic to offer various services, such as security improvement, overload control, class-based accounting, and resource reservation that require CPU resource control.

1 Introduction

Information appliances [5] are important elements to realize the ubiquitous computing vision [3]. Nowadays, most consumer electronics appliances have computing capability in order to retrieve data from sensors, to process the data, and to control devices. The recent emergence of information appliances requires more advanced features, such as networking and GUI. Those features dramatically complicate the appliances, software systems and increase their code sizes.

Networked systems need to be prepared for security attacks through the Internet. Since we expect users to be system administrators of applications, their software systems must be more robust than personal computers. Software bugs can also cause the monopolization of CPU resources. Real-time operating systems may be vulnerable to bugs of real-time programs that easily consume the whole CPU resources. For example, a problem can happen when multimedia applications process continuous media streams. Those multimedia applications have strict timing constrains, and are given the real-time priority in general. Therefore, once they process the streams, they can easily monopolize the whole CPU resources. In such a situation, the GUI process that is usually executed on the time-sharing scheduler cannot consume necessary CPU capacity, so that a user cannot control the multimedia applications through the GUI buttons. Another example is an overload condition. If an overload condition occurs, the response time of the system becomes worse.

General embedded operating systems have not yet provided a resource protection mechanism that aims to protect the CPU resources from such monopolizations and overload situations. Therefore, it is necessary to offer a generic mechanism to restrict the use of CPU resources to develop reliable information appliances. In this paper, we propose Accounting System, a general-purpose resource monitoring and restriction system that prevents the excessive use of the CPU capacity of a process or a group of processes.
The following two points are our design principles.

– *Simple design* to be applied to various services. Accounting System focuses on providing a simple and generic model and interface. Therefore, the system should be able to be easily applied to various kinds of services.
– *Accurate resource management* by using a fine-grain resolution timer. Future information appliances require to support a fine-grained rate based execution that can be realized by the accurate resource management, for offering better response time and more stable execution.

Information appliances have become very complex. Most of them nowadays contain web browsers, Java Virtual Machines, and many other applications. Implementing these applications in a robust way needs a more powerful operating system. Linux is an open-source operating system, and supports various CPU architectures. Those features are very suitable for information appliances. Therefore, many industries consider adopting Linux for their products. While the use of Linux is increasing, it does not restrict the resource consumption for their processes. We assume Accounting System can solve this problem, and implement it in the Linux kernel.

The remaining of the paper is structured as follows. Section 2 describes the design and model of Accounting System. Section 3 presents the architecture and implementation. Section 4 shows the services that can be implemented by Accounting System, such as the secure resource protection, class-based accounting, and overload monitoring. Section 5 presents the evaluation results. Finally, in Section 6, we conclude the paper.

2 Accounting System Model

Accounting System and Accounting Object: Accounting System is a system to offer the functions that monitor and restrict the execution time of a process or a group of processes. We have developed it as a part of the kernel services. Accounting System provides an abstraction to manipulate the CPU capacity of each process named *accounting object*. An accounting object represents a capacity of the CPU resource on a single processor. The CPU time of each process bound to an accounting object is accumulated in it. Through an accounting object, a user can control the allocation of CPU resources for each accounting object. When a user binds a process to an accounting object through the system call, the process is given the restricted execution time specified by its accounting object.

Time Management Model: To realize the restriction, an accounting object has two parameters C and T, where T represents a period that is a constant period to control the object, and C is the maximum time to be able to execute processes within T. The process bound to the accounting object cannot consume the CPU time more than C within T. When processes consume the entire CPU time within each period T, the process is blocked until the next period comes.

3 Implementation

3.1 Callback hooks

In Accounting System, callback hooks are introduced into the process management system in the host kernel for the purpose of catching events about processes.

Those hooks catch the relevant events, and deliver those events to Accounting System. On our design principle, Accounting System should be independent of the host kernel, and the only minimum points in the host kernel should be modified. Accounting System assumes that the four callback hooks as described next are inserted in the host kernel.

- *schedule_hook:* It catches scheduling events on context switching among processes. The hook delivers the scheduling events to Accounting System that changes the state of an accounting object.
- *fork_hook:* It catches events when a new process is created. These events are required to start the accounting for the newly created processes. If a parent process has already been bound to an accounting object, its child processes inherited the same accounting object.
- *exit_hook:* It catches events on process termination. The events notify the timing to unbind the termination of a process from its accounting object.
- *kernel_callback_hook:* It catches events when a system call returns from the kernel mode to the user mode. Accounting System checks the state of an accounting object for the blocking of the process to execute the system call.

3.2 Timer Management System

The management of timers has an important role of Accounting System. In Accounting System, there are two types of timers, a replenishment timer and an enforcement timer. A replenishment timer is a periodic timer for each accounting object that is constantly expired. An enforcement timer keeps to watch on the elapsed time of the currently executing process. They are implemented by using kernel software timers. In the following paragraph, we describe the details of them.

- *Replenishment timer:* It manages the periodic time of an accounting object. Each accounting object has its own replenishment timer. When an accounting object is created, the timer is also created. In the replenishment timer, first the expiration time is set by adding the periodic time to the current time. If the elapsed time reaches the expiration time, the registered timer handler is executed to set the next expiration time. These replenishments are recursively done by the replenishment timer until it is destroyed.
- *Enforcement timer:* It watches the elapsed time of the currently executing process. The timer accumulates the execution time of the process, until it reaches to the maximum execution time of the accounting object. At that time, the registered timer handler is invoked to change the state of the accounting object. By using schedule_hook, the enforce timer catches the events about context switching among processes, and stops and starts its timer to monitor the CPU usage of processes.

3.3 Account Object Management Operations

To manage accounting objects, the system provides the accounting object management function. It introduces the following accounting object's basic operations. To receive the accounting object services, an application firstly invokes the *create* operation to allocate the accounting object in the kernel memory.

Each created accounting object has a unique object ID. To bind a process to the specified accounting object with the object ID, the application invokes the *bind* operation. When the application no longer requires the accounting services, it invokes the *destroy* operation to terminate the accounting object service and to free the object memory in the kernel space. The *set* operation changes the parameters of an accounting object. The *get* operation obtains the parameter in an accounting object.

3.4 State Management

There are three states to represent an accounting object's status. *null* status represents the status that an accounting object is created, but no process is bound to the accounting object. *running* status represents the accounting object is running, and the status also represents that available time are left within the period time. *depleted* status represents the status that no more available time is left within the period time. The *depleted* status also represents that the accumulated execution time of a process is reached to the maximum execution time specified in the accounting object.

The state of an accounting object is changed by events. At first, when a process binds to a specific accounting object, the state becomes *running*. If the sum of the execution time of the bound processes equals to the maximum execution time of the accounting object, the object changes its state to *depleted*. Accounting System here to take action to stop their execution or deliver the notice to the bound process which is on the excessive use of the CPU capacity. The actions are classified into *block* and *signal*. If it is *block*, the system puts the processes to the wait queue. When the next expiration time comes, the replenish timer checks the wait queue, and if it finds the processes in the wait queue, sets the *running* state to the accounting object, and wakes up the processes. When a process uses excessive CPU capacity, a signal is delivered to the process. Using the state transition mechanism, Accounting System restricts the CPU usage of the processes accurately.

3.5 High Resolution Timer

For Accounting System, the high-precision clock and timestamp counter are essential to maintain the fidelity of its accuracy. To provide less than 10ms resolution for monitoring, there are some approaches. The most straightforward way is to make the clock interrupt frequency higher than 100Hz in order to issue more interrupts. This allows the kernel to provide a precise timing and to increase the system responsiveness. However, shorter intervals require CPU to spend a longer time in the kernel mode. Therefore, user programs run slower because of the system overhead to handle interrupts. Our system solves the problem by using the high resolution timer [2]. The high-resolution timer in Intel x86, is implemented by using the one shot mode of the APIC clock timer chip. Therefore, the extra timer interrupt overhead will never happen. The high-resolution timer allows us to specify microsecond granularity parameters in accounting objects.

4 Services Using Accounting System

4.1 Secure Resource Protection

Accounting System protects the CPU resource from downloaded programs that behave maliciously. We assume that there is a manager process that starts processes to execute downloaded programs. The manager process is bound to an accounting object, of which C and T are 50ms and 100ms, respectively. When the manager process receives a request to execute a downloaded application, it creates a new process. The process is automatically bound to the accounting object to which the manager process is bound. Therefore, the newly created process cannot consume more than 50% of CPU capacity, and it cannot monopolize the CPU resource.

Binding Multiple processes: Accounting System provides the *bind* operation that binds a process to the accounting object. We also provide some useful interfaces to bind multiple processes to an accounting object. For example, if a malicious program consecutively invokes the *fork* system call, the whole system resources may be consumed. We implement the following two methods. The one is to inherit an accounting object when creating a child process. The other is to bind an accounting object using group ID. The former function is realized by using a hook within the *fork* system call. If a parent process is bound to an accounting object, the child process who is forked from the parent inherits the same accounting object. Thus, the same restriction as its parent is applied. The latter uses UNIX group process ID. The system binds the group of processes to a specific accounting object. It is safer than binding all processes to the same accounting object respectively. For example, when several commands connected by pipes are started, they can be easily bound to the same accounting object by using group ID.

Access Control: Each accounting object has an owner attribute for controlling the access to a specific accounting object. The owner attribute of the accounting object is assigned by a process that creates it. For the consideration of the security, our system allows them to be manipulated only by their owners except the privileged user. In resources reservation systems such as Linux/RK [4], there is no support of access control. Therefore, these previous system cannot be used for protecting the CPU resource from malicious programs.

Kernel Interface: The following is the kernel interface offerd by Account System.

- *account_create(&object_id,&object_attr)* It creates a new accounting object. When created, the object is not bound to any processes. A privileged or owner application program sets the parameters, such as C, T, in the object_attr memebers.
- *account_destroy(&object_id)* It destroys the accounting object specified by object_id. This deletes the associated timers, and frees the memory of the accounting object in the kernel address space.

- *account_bind/unbind_pid(object_id, pid_t)* It binds/unbinds the specified accounting object through object_id to a process whose process ID is pid. An accounting object can be bound to multiple processes. If the process is terminated, the process is automatically unbound from the accounting object.
- *account_get/set(object_id, &object_attr)* It retrieves/changes the parameters of the specified accounting object.

4.2 Class-based Accounting

In this section, we propose a class-based accounting as the one of the services that uses Accounting System. A class is defined here as a group of processes that belong to a specific scheduling class, such as the real-time scheduling class. The class-based accounting is realized by binding the processes belonging to the same scheduling class.

A problem of process starvation: When a user forgets to bind a real-time process to an accounting object, the real-time process may monopolize the entire CPU resource. Even if most real-time processes are restricted, the only one real-time process can consume the rest of all the CPU resources; thus, all other time-sharing processes are starved. To avoid such a situation, the system designer should estimate the total maximum utilization of the accounting objects, which bind to the all real-time processes. We called the approach class-based accounting.

Class-based Accounting: The class-based accounting is realized by binding processes in the specific scheduling class to an accounting object. A class contains all processes that are scheduled by the same scheduling policy. The system allocates the CPU resources proportionally to each scheduling class. The total proportions of the classes are set to 100%. For example, a user can allocate 40% of the CPU capacity for the real-time scheduling class, and the remaining 60% of the CPU capacity for the time-sharing scheduling class. The system assumes to have two classes that are necessary to allocate the absolute rate of the CPU capacity.

The processes bound to each class cannot excessively use the maximum utilization of the CPU resources that are set in each scheduling class. Since the real-time process cannot use more than the assigned capacity, the time-sharing process can use the remaining capacity without process starvation.

Implementation: The current class-based accounting allocates the CPU capacity for the real-time and time-sharing scheduling class. The accounting object for each scheduling class is created at the boot time. When it is created, all processes that belong to the same scheduling class are bound to the same accounting object. In order to set the parameter easily, class-based accounting adopts weight as its interface parameters. The weight is the parameter that represents a proportion and can easily be specified like (2,3). If weight (2,3) given, the real-time class and the time-sharing class are allocated 40% and 60% CPU capacity respectively. In class-based accounting, the accounting object of each scheduling class should use the same period time. Also, a user needs to set the resolution

time that is a period for the accounting objects. For example, if the resolution is 100ms and the class proportion is 40%, the class is received 40ms within the 100ms period.

Kernel Interface: We design the kernel interface for class-based accounting as described as following.

- *weight_set (wt_ts, wt_rt, res)* Sets the weight and resolution to the accounting object for respective scheduling classes. When the class accounting object is started, it monitors the execution time of the bound processes.

4.3 Overload Monitoring

Conceiving the overload situation: For general-purpose operating systems, controlling overload situations is very important to offer services in a stable way. In such critical situations, the computational requests from the process will exceed the time that is available in the system. As for time-sharing processes, the average response time is increasing. We propose the monitoring function that keeps to watch the total utilization of the CPU. If the overload condition happens, the overload monitoring system issues a signal to notify to the registered administrator process. If there is a notification for the administrator process, some recovery procedures can be taken to stabilize the system.

Design and Implementation: The overload monitoring can also be realized by Accounting System. In this system, an accounting object bound to the idle process is used to monitor the total utilization of the system resources. The idle process is executed when there is no runnable process. If the system becomes busy, the idle process consumes a very little CPU capacity. Through monitoring the CPU usage of the idle processes, the system can detect the overload situation. Our system allows an application to receive a signal when the CPU usage of the idle process becomes lower than the specified CPU. The service is available when a system designer creates an overload accounting object, and sets the parameters to it. There is the only one overload accounting object in a system. A system designer should explicitly set the C and T parameter to the overload accounting object. Suppose the designer sets C and T to 75ms and 100ms respectively, it monitors the CPU capacity not to exceed 75%. If the designer would like to check the overload condition every minute, C and T should be set to 45sec and 60sec respectively. If the total usage of CPU utilization exceeds 75%, an overload notification signal is delivered to a specific process. A system designer also should set a process as a receiver of the signal. The signal number can be freely decided by a user. In consideration of the security, only a privileged user can create and destroy the overload accounting object.

Kernel Interface: The overload monitoring offers the kernel interface as described as follows.

- *overload_create (&object_attr, pid)* The function binds an idle process to the overload accounting object. The user sets an administrator process ID that accepts a signal from the system and a signal number that is used to deliver the signal to the administrator process when an overload condition occurs.

– *overload_destroy(void)* It destroys the overload accounting object.

4.4 Resource Reservation

An application running on our system can request the reservation of a certain amount of CPU resources. The OS-enforced resource reservation can be introduced similar to CPU capacity reserves [1] [4], which provide the framework for managing the processor capacity. Accounting System can implement the framework easily by adding a mechanism for the admission control as a user-level service.

5 Evaluation

We have evaluated Accounting System by running several benchmarks. The evaluation uses a standard PC that has Celeron 300MHz CPU and 512MB of RAM, and the results were measured using the built-in high-resolution timestamp counter.

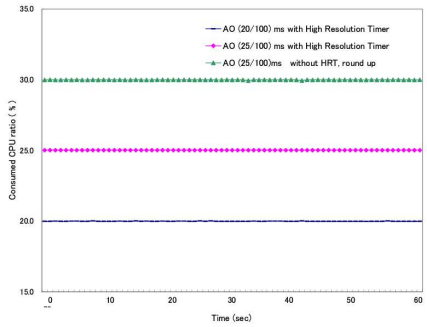

Fig. 1. With/Without High Resolution Timer Support

5.1 Accuracy of Accounting System

This section evaluates the accuracy of Accounting System. To evaluate the effectiveness of high-resolution timer support, we use the benchmark program that executes the infinite-loop. We create two accounting objects that set 20ms and 25ms of CPU time every 100ms, and bind the program to each accounting object. Figure 1 shows the result of the consumed CPU ratio with/without the support of a high-resolution timer. In this figure, the bottom two lines are the results of the high-resolution timer support. It shows that each 20% and 25% resource restriction is correctly limited. In the contrast, without the high-resolution timer support, the program bound to the accounting object whose execution time is 25ms executes 30ms within 100ms. These are apparently due to the fact that Linux updates its internal timer-clock at the 10ms intervals. Even if a user sets a parameter whose resolution is less than 10ms, it is impossible to ensure the resolution. The results shows that the effectiveness of the accurate resource accounting.

5.2 Effectiveness of CPU Protection

This section evaluates the effectiveness of the CPU resource protection in Accounting System. First, we show the effect of the security attacks, then, show the result of the class-based accounting.

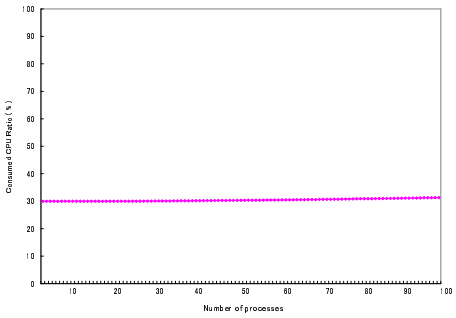

Fig. 2. The Numbers of Process and Accounting Object Ratio

Attacks to create multiple processes: Figure 2 shows the consumed CPU ratio of multiple processes bound to an accounting object. To evaluate the effectiveness for protecting from attacks to create multiple processes, we run an evil program that invokes fork() consecutively and produces a lot of child processes that execute their infinite-loops. The response time becomes very bad by this malicious program because these processes consume the whole CPU resource. In the evaluation, if the first process is bound to the 30% restricted accounting object, the forked processes cannot use more than 30% of the CPU capacity, even if the numbers of processes are increasing. The result shows the effectiveness of Accounting System as shown in Figure 2. The loss of the accuracy appears as the numbers of created process is increased. In the current implementation, a process that executes a system call is blocked after the system call is returned, when the execution time of the accounting object is expired. If there are multiple processes that are bound to the same accounting object, and they execute system calls, the system calls can be executed before blocking themselves. This is a reason to exceed the specified execution time when the number of process bound to the same accounting object is increased.

Class-based Accounting: Figure 3 and Figure 4 show the results of the evaluation of class-based accounting. To evaluate the effectiveness of the class-based accounting, the benchmark runs a real-time process to execute an infinite loop with/without the class-based accounting. Figure 3 shows the result when the class-based accounting is not used. The result shows that all CPU resources are consumed by the real-time process. Then, we evaluate the class-based accounting in which each class is set the weight (4,1) where 80% and 20% CPU capacity are allocated for real-time processes and time-sharing processes respectively. The

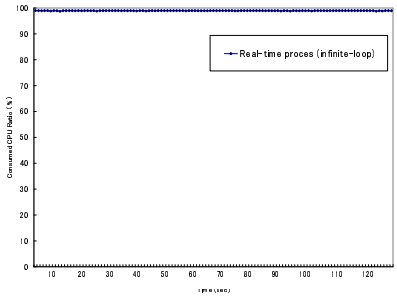

Fig. 3. Without Class-Based Accounting

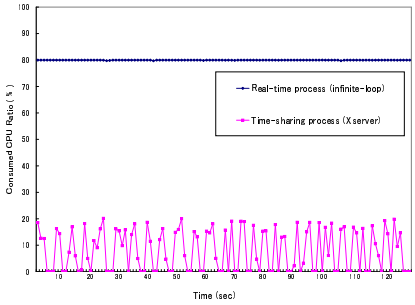

Fig. 4. With Class-Based Accounting

Xserver (X Window System display server) process runs on the time-sharing scheduler. On Xserver, some game programs are running. Figure 4 shows the result that the real-time processes can not exceed 80% CPU resources, where the Xserver process can use 20% CPU resource at a maximum.

6 Conclusion

In this paper, we have proposed Accounting System. This system provides the generic accounting model that can be applied various services that are used to increase the system security and reliability. It also supports a fine-grained resource accounting with the high-resolution timer. This makes the system more stable because the application can obtain the necessary rate precisely even in the microsecond resolution. The performance evaluation results have showed that the system effectively protects the malicious use of CPU resources.

References

1. Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda.: Processor capacity reserves: Operating system support for multimedia applications. In Proceedings of the IEEE Internation Conference on Multimedia Computing and Systems, pages 90–99, May 1994.
2. High Resolution Timer, http://high-res-timers.sourceforge.net/
3. Mark Weiser.: The Computer for the 21st Centriy, Scientific American, Vol 265, No.3, 1991.
4. Shuichi Oikawa and Ragunathan Rajkumar.: Portable RK: A portable resource kernel for guaranteed and enforced timing behavior. In Proc. IEEE RTAS, 1999.
5. W. P. Sharpe, S. P. Stenton.: Information Appliances, The Human-Computer Interaction Handbook : Fundamentals, Evolving Technologies and Emerging Applications - Human Factors and Ergonomics, Chapter37, Lawrence Erlbaum Assoc Inc Published 2002.