Starvation Problem in CPU Scheduling For Multimedia Systems

M. O. Saliu, K. Salah*
Department of Information and Computer Science
King Fahd University of Petroleum & Minerals
KFUPM # 1067, Dhahran 31261, Saudi Arabia.
Email: {saliu, salah}@ccse.kfupm.edu.sa

Abstract

One of the major tasks of traditional general-purpose operating system is to provide an orderly and controlled allocation of processor among various executing programs competing for it in a fair and efficient manner. Multimedia applications have timing requirements that cannot generally be satisfied using the time-sharing algorithms of general-purpose operating systems. Integrating discrete and continuous data of digital audio and video requires additional services from operating systems, especially handling of time-constrained characteristics of continuous media data, which poses a real-time characteristics on the underlying scheduler. Implementing multimedia applications using a real-time scheduler leads to starvation of conventional applications. In this paper, we briefly describe three of the popular multimedia scheduling algorithms. We compare and discuss how adequate each algorithm is in handling the issue of starvation. Additionally, we propose a new improvement for handling starvation for one of the most popular multimedia scheduling algorithms.

Keywords: Scheduling, Real-time, Multimedia, Starvation

1 Introduction

Multimedia application refers to the capture, storage, retrieval and presentation of audio and video data using computers. Audio and video data streams consist of periodically changing values of continuous media data such as audio samples and video frames, and these convey

* Corresponding Author

appropriate meaning only when presented continuously in time. Multimedia applications handling audio and video data have to obey time characteristics of these media types, and are normally classified as soft real-time applications, because of their requirement for timely correct behaviors.

Traditional real–time scheduling techniques used for control systems in application areas such as aircraft piloting, demand high security and fault tolerance. The fault tolerance requirements of multimedia systems are usually less strict. Short-time failure of a continuous media system, such as deadline misses do not lead to catastrophic consequences, but could degrade the *Quality of Service (QoS)*. It may even go unnoticed.

The goals of traditional scheduling on general-purpose operating systems, like UNIX and Windows NT, are to provide optimal throughput, optimal resource utilization and fairness. In contrast the main goal of real-time tasks is to provide a schedule that allows for as many time-critical processes as possible to be processed in time to meet their deadlines. None of the existing general-purpose operating system has been designed to provide multimedia data processing support. For the scheduling of multimedia tasks, therefore, the following objectives have to be considered:

- Time-critical tasks have to be scheduled so that they can always meet their execution deadlines. This calls for real-time scheduling policies.
- Starvation of non-critical processes (such as those required to keep the system running), due to the execution of time-critical tasks, is unacceptable. Since this may be in conflict with the previous objective, it is necessary to realize systems where time-critical and non-critical tasks can co-exist.

Starvation is an issue that should be well addressed in any scheduling algorithm for multimedia systems if a reasonable level of quality-of-service is desired. Comparison and evaluation of the basic approaches employed by different scheduling algorithms for scheduling multimedia systems form the crux of the contributions are made in this paper.

The rest of this paper is organized as follows: Section 2 presents our problem statement, and Section 3 discusses some multimedia scheduling schemes and the rationale behind their various algorithms which form the basis for our comparison. Evaluation of the scheduling algorithms in terms of starvation and their drawbacks as compared to other schemes is done in Section 4. Section 5 proposes an improvement to one of the popular scheduling algorithms to handle starvation. Finally, Section 6 contains our conclusion and further study.

2 Problem Statement

Our motivation to study multimedia applications scheduling algorithms derives from the conflicting objectives stated in the previous section. In this paper, we will compare some of the scheduling algorithms for multimedia in the literature, and how the issue of starvation is handled to take care of requirements imposed by various applications that may co-exist in a multimedia system.

In [BAV00], three multimedia algorithms SMART [NIE97], BERT [BAV99] and BVT [DUD99] were compared based on their implementation of *virtual time.* Our comparison, on the other hand shifts towards starvation issue, whether handled or sacrificed, for some popular multimedia scheduling algorithms.

3 Popular Multimedia Scheduling Algorithms

Some of the early works in the area of OS support for multimedia systems focused on the real-time aspect, and classical approaches for real-time processing like, earliest-deadline first (EDF), rate monotonic (RM) and least-laxity first (LLF) were adopted [STE95]. A danger of priority scheduling like earliest-deadline first is starvation, in which processes with lower priorities are not given the opportunity to run. Although they are appropriate for hard real-time applications, yet these algorithms are not suitable for soft real-time multimedia, and conventional applications.

Several new algorithms have recently been developed, some of which still employ one or two of those mentioned above. They can be classified as *proportional share resource allocation, reservation-based, and hierarchical algorithms* [PLA00]. Proportional share schedulers are quantum-based weighted round-robin that guarantees that an application with N shares (or weights) will be given at least N/T of the processor time, on average, where T is the total number of shares over all allocations [REG00]. Some of the approaches employing proportional share include *scheduler for multimedia and real-time applications (SMART)* [NIE97], *borrowed-virtual-time* (*BVT)* [DUD99] and *adaptive rate-controlled (ARC)* [YAU96]. In reservation-based, an application is provided with load isolation (reserving an amount of CPU per period) and periodic execution. One of the algorithms that belong to this category is *processor capacity reserves in Real-Time Mach* [MER94]. *Hierarchical* algorithm generalizes the traditional role of schedulers by allowing them to allocate CPU time to other schedulers. Some examples of schedulers in this group are *hierarchical start-time fair queuing (SFQ)* [GOY96], and *soft real-time (SRT)* [CHU99].

3.1 Processor Capacity Reserves

This approach provides a scheduling mechanism that allows users to control allocation of processor cycles among programs. Applications request processor capacity reservations, and once a reservation has been granted by the scheduler, the application is assured of the availability of processor capacity. It uses admission control to allow real time tasks to reserve a fixed percentage of the resources in meeting real-time requirements. The reservation system was designed to support higher-level resource management policies; for example, a quality of service (QoS) manager could use the reservation system as a mechanism for controlling the resources allocated to various applications, by translating their QoS parameters to system requirements.

In this algorithm, programs are scheduled consistently with the admission control policy, and an accurate measure of the computation time consumed by each program is done to ensure that programs do not overrun their reservations, hence interfering with other programs.

Processor percentage provides a means of measuring requirements of both time-constraint and non-time-constraints applications, and the processor percentage consumed by a program over time defines its rate of progress. The Rate Monotonic and Earliest-Deadline First algorithms are suitable for implementing reservation scheduling in the framework, because they are methods of assigning priorities to programs which ensure that each program progresses at its assigned rate.

Using rate monotonic for fixed priority and given that n is the number of periodic programs, while C_i and T_i represent the computation time and period of program i respectively, an admission control policy follows from the rate monotonic scheduling analysis from [LIU73] that;

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right) \qquad (1)$$

When n is large, $n\left(2^{1/n} - 1\right) = \quad ln\ 2 \approx 0.69$

The reserved rates of programs that have been admitted are recorded, and the total reservation is the sum of these rates. A simple admission control policy is to admit a new program if the sum of its rate and the total previous reservation is less than 69% using rate monotonic. With earliest-deadline first for dynamic priority policy, however,

$$\sum_{i=1}^{n} \frac{C_i}{T_i} = 1 \qquad (2)$$

an admission policy here admits if sum of rates is less than 100%, otherwise the reservation cannot be granted. The assumption is based on [LIU73] assumptions that all programs will successfully meet their deadlines and compute at their associated rates, given that (1) and (2) hold for rate monotonic and earliest- deadline first respectively. Programs that have not yet consumed their reservation take precedence over unreserved processor time available, but if there is unreserved processor time available, unreserved programs can take advantage of the extra processor time.

3.2 Hierarchical Start Time Fair Queuing (SFQ)

SFQ has a CPU allocation framework suitable for multimedia operating system. The framework enables different schedulers to be employed for different application classes, and a tree specifies the hierarchical partitioning. Each thread (task) belongs to exactly one leaf node and each node represents an application class. Threads are scheduled by leaf node schedulers while intermediate nodes are scheduled by an algorithm that achieves hierarchical partitioning – SFQ.

If an application requests a real-time service, the QoS manager uses admission control algorithm which utilizes the capacity allocated to the class to determine if the request can be satisfied, and if so, assigns it to the appropriate partition. If application requests best-effort service, then the request is not denied but assign to appropriate partition. Each node has a weight that determines percentage of its parent node's bandwidth that should be allocated to it. Figure 1 shows a scheduling structure where the three sub-classes: hard real-time, soft real-time and best-effort, has weights 1,3, and 6 respectively, with further sub-division of the best-effort class equally among leaf classes. While soft real-time and user1 leaf classes employ a fair scheduler, the hard real-time and user2 classes have EDF and time-sharing schedulers, respectively.

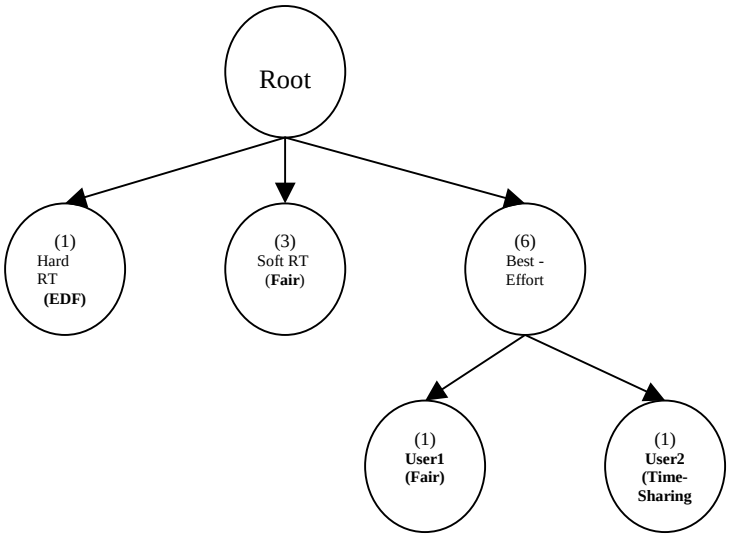

Figure 1: An example scheduling structure.

In SFQ, each competing thread should receive an amount of CPU bandwidth that is proportional to its weight during any arbitrary finite interval. The number of *normalized work units* allocated to a thread is defined to be the ratio between the numbers of allocated CPU instructions and the thread's weight. The proposed scheduling algorithm approximates fair scheduling by monitoring

the number of normalized work units that were allocated to each process, and give higher priority to processes that were given less than their fair share of the CPU bandwidth.

The scheduler uses the notion of virtual time to do scheduling. Virtual time starts at 0 and is constant during the execution of a time quantum. Virtual time is expressed in terms of normalized work units. When the CPU is busy, the virtual time is equal to the start tag of the thread in service at that time. On the other hand, when the CPU is idle, the virtual time is set to the maximum of finish tag assigned to any thread. Each thread has a start tag and an end tag associated with it. Threads are serviced in the increasing order of the start tags, and ties are broken arbitrarily. To have a flavor of how fair scheduling is achieved with SFQ algorithm, an example in *[GOY96]* gives a detail description.

3.3 SMART: A Scheduler for Multimedia And Real-Time Applications

SMART reduces the complex resource management problem into two decisions, one based on *importance* to determine the overall resource allocation for each task, and the other based on *urgency* to determine when each task is given its allocation. SMART provides a common importance attribute for both real-time and conventional tasks based on priorities and weighted fair queuing (WFQ). SMART then uses an urgency mechanism based on earliest-deadline scheduling to optimize the order in which tasks are serviced to allow real-time tasks to make the most efficient use of their resource allocations to meet their time constraints. A bias on conventional tasks that accounts for their ability to tolerate more varied service latencies is used to give interactive and real-time tasks better performance during periods of transient overload.

An urgent task is one that has an immediate real-time constraint. On the other hand an important task is one with a high priority, or one that has been the least serviced proportionally among applications with the same priority. An urgent task may not be the one to execute if it requests more resources than its fair share. Conversely, an important task needs not be run immediately. For example, a real-time task that has a higher priority but a later deadline may be able to tolerate the execution of a lower priority task with an earlier deadline. SMART avoids a situation where a real-time process that is not of much importance takes over the resource from an important conventional application.

The importance of an application is measured by a *value-tuple,* which consists two components: *priority* and the *biased virtual finishing time* (*BVFT*). Priority is a static quantity either supplied by the user or assigned the default value. *Virtual finishing time* (VFT) is a dynamic quantity the system uses to measure the degree to which each task has been allotted its proportional share of resources, and it incorporates tasks with different priorities. A task *A* has a higher value-tuple than task *B* if *A* has a higher static priority or if both *A* and *B* have the same priority and *A* has an earlier BVFT.

The SMART scheduling algorithm used to determine the next task to run is as follows:

1. If the task with the highest value-tuple is a conventional task (a task without a deadline), the task is scheduled.
2. Otherwise, a candidate set is created consisting of all real-time tasks with higher value-tuple than that of the highest value-tuple conventional task.

3. The best-effort real-time scheduling algorithm is applied on the candidate set, using the value-tuple as the priority. The algorithm attempts to schedule the candidate into a working schedule, which defines execution order. The candidate is inserted in increasing deadline order in this schedule provided its execution does not cause any of the tasks in the schedule with higher value-tuple to miss its deadline. The scheduler simply picks the task with the earliest deadline in the working schedule.
4. If a task cannot complete its computation before its deadline, a notification is sent to the application.

SMART behaves like a real-time scheduler when scheduling only real-time requests and behaves like a conventional scheduler when scheduling only conventional requests. However, it combines these two dimensions in a dynamically integrated way that fully accounts for real-time requirements.

4 Comparative Evaluation of Starvation Issues in Selected Algorithms

In this section, we compare and discuss how adequate each scheduler in handling the issue of starvation.

Processor Capacity Reserves clearly sacrifices fairness in its framework; this makes starvation of conventional applications inevitable. This is evident from the reservation policy implemented that allows real-time programs to reserve a fixed percentage of the resource in accordance with their resource requirements, and an application of real-time scheduling to execute real-time tasks, which results in starvation of conventional tasks. Any leftover processing time is allocated to

conventional tasks using a standard timesharing or round-robin scheduler. In spite of the fact that reservations are targeted towards giving real-time systems better performance, late arrival of a real-time application in an overloaded system can result in its denial by the admission control system, even if it is more important than already admitted processes.

H*ierarchical Start-time Fair Queuing Scheduler (SFQ)* implements fair sharing, but synchronization between threads can lead to priority inversions that would destroy the fairness of the scheduling algorithm, and hence leading to starvation. Since it separates scheduling policy for real-time and conventional applications, it is limited by combination mechanism, thus a thread from real-time class synchronizing with another thread in the best-effort class (which does not perform any admission control), may violate the quality of service requirement of the thread. In particular, the algorithm does not extend scheduling decision to the lowest-level where the actual scheduling of processor cycle is done. The decision is left in its entirety to the leaf scheduler. Also, real-time applications will not take over the machine, but they also cannot effectively meet their time constraints as a result of the underlying proportional share mechanism taking the resource away from the real-time scheduler at an inappropriate and unexpected time in the name of fairness. Meanwhile, there is no provision for feedback to allow real-time applications to adapt effectively in such situation.

SMART tolerates some instantaneous unfairness, so as to meet deadlines and deliver good response time to short-running tasks. And depending upon priorities, new time constraints can steal time that might otherwise have been needed to finish an existing constraint on time, or to maintain other applications proportional share requirements. Thus it provides no guarantee for

real-time tasks unless they are executed at the highest priority, and there is possibility of starving time-sharing tasks. As it is generally identified with priority-based systems, and with *SMART* using priority as the first major criteria for scheduling, a system that is composed of basically high-priority real-time applications leaves an open room for the starvation of conventional applications, since they can not be scheduled alternatively using urgency. This implies that the admittance of real-time processes should be managed. The SMART adaptability feature allows for already admitted processes to manage their own degradation policies in the event that enough CPU resource is not available. However, it does not provide admittance in an overload situation. This situation still can clearly result in starvation of applications whether of low or high priority.

5 Suggestions for Improvement

From our comparison in the previous section, we discovered that *SMART* needs some more features to effectively handle starvation, in spite of its numerous features. We suggest that admission control policy be clearly implemented in *SMART*. An algorithm does not necessarily need to make reservation in order to control admittance of applications, as claimed in [NIE97]. In this case, therefore, an evaluation of resource requirements of a real-time task can be done against available resources so as to decide on its admission. With admission control, *SMART* will be able to evaluate the timing constraints of new programs against the available processor capacity. Hence, an immediate feedback is given to the application so it can adjust its timing requirements accordingly, rather than executing to a point and have the other computation discarded. An uncontrolled new real-time application of the same priority level with an equally important conventional application will take over processor resource since it has not accumulated

any bias. Dependencies among various application classes should also be reduced, so that the fate of conventional process does not completely rely on the priority of real-time processes. This is not intended to mean that real-time characteristics of multimedia applications be sacrificed.

6 Conclusion and Further Study

In this paper, we have investigated some of the popular scheduling algorithms for multimedia operating systems, and discussed their handling of starvation in a system containing application mix of both soft real-time multimedia and time-sharing conventional systems.

We conclude that *SMART,* is a very important scheduling algorithm since its policies are directly propagated to the lowest scheduling level while giving proper attention to different classes of applications. On the other hand, *Hierarchical* algorithm relies on some other scheduler, the leaf scheduler, to complete process scheduling. Apart from being a scheduler on its own, *SMART* can also be used as a leaf scheduler in hierarchical system, as well as scheduling multimedia in multiprocessor environment as implemented in [NIE98]. With this in mind, we suggested an approach that incorporates admission control in *SMART* so as to avoid situations where a high priority real-time process takes over total control of the system, thereby leading to starvation of other low-priority real-time and conventional applications. For further study, a simulation and an implementation of the proposed improvement for *SMART* are underway to validate our claims.

References:

[BAV00] Bavier A., Peterson L.: ”The Power of Virtual Time for Multimedia Scheduling”. Proceedings of NOSSDAV2000, JUNE 2000.

[BAV99] Bavier A., Peterson L., Mosberger D.: ”BERT: A scheduler for Best Effort and Realtime Tasks”. Technical Report TR-602-99, Department of Computer Science, Princeton University, March 1999.

[CHU99] Chu H., Nahrstedt, K.: “CPU Service Classes for Multimedia Applications”. Proc. of IEEE Int. Conf. On Multimedia Computing and Systems (ICMCS’99), Florence, Italy, June 1999.

[DUD99] Duda K.J., Cheriton D.R.: “Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general purpose scheduler”. In proceedings of the 17th ACM Symposium on Operating System Principles, Dec. 1999.

[LIU73] Liu C.L., Layland, J.W.: “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment”, Journal of the ACM, Vol. 20, No. 1, January 1973, pp. 46-61.

[MER94] Clifford W. M., Stefan Savage, and Hideyuki Tokuda

"Processor Capacity Reserves: Operating System Support for Multimedia Applications".

Proceedings of the IEEE International Conference on Multimedia Computing and Systems, May 1994.

[NIE98] Nieh J., Lam, M.S.: “Multimedia on Multiprocessors: Where’s the OS When You Really Need It?” In Proc. of the 8th International Workshop on Network and Operating System Support for Digital Audio and Video, Cambridge,U.K., July 1998.

[NIE97] Nieh J., Lam, M.S.: “The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications”, Proc. of 16 th ACM Symp. on Operating System Principles (SOSP’97), St. Malo, France, October 1997, pp. 184-197.

[PLA00] Plagemann T., Goebel, V., Halvorsen, P., Anshus, O.: "Operating System Support for Multimedia Systems", The Computer Communications Journal, Elsevier, Vol. 23, No. 3, February 2000, pp. 267-289.

[REG00] Regehr J., Jones M.B., Stankovic J.A.: “Operating System Support for Multimedia: The Programming Model Matters”. Technical report: University of Virginia CS-2000-07, March 2000.

[STE95] Steinmetz R., “Analyzing the Multimedia Operating System”, IEEE Multimedia, Spring, 1995, pp 68-84.