On Kernel Support for Real-Time Multimedia Applications

Kevin Jeffay
University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC, USA 27599-3175
jeffay@cs.unc.edu

Abstract: Real-time operating system services are required to support multimedia systems that rely heavily of the workstation processor for control of the audio and video processors and movement of audio and video data. Such services are typically not available in existing workstation operating systems. This note comments on the requirements for such services and briefly describes the YARTOS kernel; an operating system kernel that provides real-time communication and computation services.

Introduction

Recent advances in video compression algorithms — and their realization in silicon — have made it possible to consider introducing streams of digitized audio and video into the processing workload of workstation operating systems. For example, by outfitting workstations with off-the-shelf video cameras, microphones, digital video and audio acquisition and compression hardware, and audio amplifiers, it is possible to construct multimedia applications such as integrated voice/video/text documents and browsers [Hopper 90] as well as communication utilities such as workstation-based video and/or audio conferences [Terry & Swinehart 88, Jeffay & Smith 91].

While the hardware for such systems is readily available, existing operating systems and network communication protocols are inadequate for supporting multimedia applications such as browsing a video document or conferencing. This is due to the real-time processing requirements of digital audio and video, specifically, rigid throughput and latency requirements. For browsing or conferencing in a distributed system, frames of video must be acquired at a remote workstation (either from a camera or from a disk file) and transmitted so as to arrive at the local workstation and be displayed at the (precise) rate of one frame every 33 ms. Problems such as these have stimulated programs of basic research in many of the

In: Proc. Third IEEE Workshop on Workstation Operating Systems, Key Biscayne, FL, April 1992, pp. 39-46.

traditional areas of operating systems such as file systems [Rangan & Vin 91], and scheduling and inter-process communication [Anderson *et al.* 90, Govindan & Anderson 91, Jeffay *et al.* 92b].

My interest lies in the development of operating system infrastructure for the processing of *live* digital audio and video, specifically, workstation-based conferencing. Applications requiring live digital audio and video, such as conferencing, are unique in that their real-time throughput and latency requirements are particularly demanding. A conferencing application fundamentally requires that the audio and video data be processed *as it is generated* (*i.e.*, with zero or one buffer). To do otherwise implies that either portions of the conference will not be reproduced (*e.g.*, frames will be dropped) or that artificial latency is imposed between acquisition and display processes. In order for a system to be usable as a conferencing tool, one should minimize, if not avoid altogether, the occurrence of these events. Ideally a workstation-based conferencing system should be indistinguishable from the more traditional analog (*i.e.*, non-computer based) system.

At UNC we have constructed an experimental network of workstations capable of processing live digital audio and video and are using this system to experiment with operating system and network support for continuous-time media. In designing the system and application software for our network, the approach has been to view the problem as one of real-time resource allocation and control. Rather than try to add a "real-time scheduling algorithm" to existing operating systems (or resurrect real-time UNIX) I have been considering what an operating system specifically biased to providing real-time services might look like.

This brief note outlines one assessment of the salient requirements for next generation operating systems that will support multimedia (and other real-time) applications. The ideas presented here are based on experiences with the design and implementation of the YARTOS (Yet Another Real-Time Operating System) workstation operating system kernel and with the use of this kernel to construct such multimedia applications as a workstation-based conferencing system and a simple VCR program [Jeffay *et al.* 92a,b]. A brief description of the YARTOS kernel is also presented.

Real-Time Requirements for Multimedia

The extent to which multimedia applications will require real-time computation and communication services from an operating system depends in large part on the architecture of the audio and video (A/V) subsystem; in particular, the relationship between the A/V

subsystem and the CPU. At one extreme are systems wherein digital or analog video and audio signals are acquired by largely autonomous A/V processors that are capable of delivering A/V data directly to the workstation frame buffer with little, if any, intervention by the CPU. In a workstation with an autonomous A/V subsystem the CPU typically communicates with the A/V subsystem via a high-level device control interface (*e.g.*, *StartVideoStream*, *StopVideoStream*) and none of the movement of A/V data is done under direct CPU control. Examples of such systems include the direct attachment of off-the-shelf teleconferencing video CODECs to workstations [Bosco 91]. Such multimedia architectures have the advantage that they typically do not require modifications to the workstation operating system. The disadvantage of such hardware solutions is that they are expensive and of rather limited use as A/V data is typically not visible to user processes.

At the other extreme are computer systems wherein the CPU is intimately involved in the control of the A/V processors and in the communication of A/V data between the processes controlling the A/V processors. An example of such a system is the video conferencing system we have developed using IBM-Intel ActionMedia 750 products [Jeffay *et al.* 92b, Harney *et al.* 91]. In this system the CPU directs the acquisition, compression, and transmission of *each frame* of audio and video including all movement of A/V data within the system. At this time such "dependent" A/V subsystems are significantly cheaper than autonomous systems. More importantly, within a dependent A/V subsystem, A/V data is available to user processes on a frame by frame basis thereby allowing user applications to perform operations such as filtering, compositing, and authentication on A/V streams. The primary drawback of dependent A/V subsystems is that they are harder to use effectively with existing operating systems because operating systems do not support the real-time demands of these A/V devices and applications.

The present work is aimed at workstations with dependent A/V subsystems. The effective use of these A/V subsystems requires that real-time services be provided at a number of levels within the kernel. At the lowest level real-time device control is required to acquire, manipulate, and display digital audio and video. For example, for a conferencing application using ActionMedia hardware, each video frame propagates through a three stage pipeline: digitization of the video frame, compression of the digitized image, and transmission of the compressed data over the network. Each stage of the pipeline is implemented on a separate physical processor that is controlled by processes (*e.g.*, device drivers and application programs) that execute on the CPU. The timing of the execution of these processes is critical to the correct functioning of the pipeline. For example, a *single* late control signal will result in a stall and resynchronization of the pipeline and will create a noticeable "glitch" (*e.g.*, an

audible “pop”) in the outgoing A/V stream. Therefore, application and device driver processes must be scheduled so as to ensure guaranteed response times to events (*e.g.*, message arrivals, hardware interrupts). Moreover, sufficient amounts of software resources such as buffers must be guaranteed to be available to these processes when they are scheduled. These requirements are not met by existing workstation operating systems.

On Real-Time Services

While scheduling and resource allocation are clearly needed (and receive most of the attention in the real-time systems literature) a general statement of the operating system/application interface is missing. Scheduling alone is too primitive a tool to construct applications such as video conferencing that require the use of multiple, CPUs, I/O devices and controllers in concert. A kernel must provide real-time computation and communication services that enable programmers to both specify both real-time throughout and latency requirements for individual processes and to assess real-time performance issues such as end-to-end latencies. Moreover, a kernel should support a tasking model for which it is possible to determine *a priori* if sufficient processing resources are currently available to meet an application’s requirements.

While the notion of real-time services for which one provides *a priori* guarantees of performance may not be necessary for all applications, it is important to understand the cost of providing such services. For example, as indicated above, if ActionMedia processes are not scheduled in real-time then it is likely that a less than full fidelity stream of audio and video will be generated. While by some measures this may not adversely effect the use of the system, it is fundamentally up to the application designer or end user to define such measures of goodness for the performance of applications. The operating system’s explicit goal (and that of its designer) should be to accommodate these requirements whenever possible. Ideally we would like to provide a range of real-time services from “guaranteed response time” to “best effort” and and allow application programmers to decide if the cost of the real-time service is worth its perceived benefit. In our experience the “cost” of hard-real-time services typically takes the form of a restricted programming model for real-time processes and possibly degraded performance for non-real-time processes.

Moreover, beyond the desire for predictable computation services, it is important to understand the cost of real-time computing in order to effectively trade-off computing resources for application performance requirements when the computing resources are scarce. It is quite easy to design real-time applications that saturate available computing resources. This is especially true for current multimedia systems when users desire to interact with

multiple streams of digital audio and video simultaneously. It is difficult, if not impossible to effectively ameliorate the effects of saturation without a good understanding of how application performance requirements would be met on systems with theoretically sufficient processing capacity.

Kernel Support for Predictable Real-Time Computing

The YARTOS kernel was developed to experiment with the design and implementation of real-time communication and computation services. These services are realized through a paradigm of process interaction called the *real-time producer/consumer (RTP/C) paradigm* [Jeffay 89]. The RTP/C paradigm defines a semantics of inter-process communication that provides a framework for reasoning about the real-time behavior of programs. This semantics is realized through an application of some recent results in the theory of deterministic scheduling and resource allocation. The claim is that YARTOS is a "general purpose" real-time operating system kernel. (In addition to the conferencing application, YARTOS prototypes have been used in a 3-dimensional interactive graphics system used for research in *virtual realities*, and a HiPPI data link controller.)

The programming model supported by YARTOS is an extension of Wirth's discipline of real-time programming [Wirth 77]. In essence it is a message passing system with a semantics of inter-process communication that specifies the real-time response that an operating system must provide to a message receiver. This allows a programmer to assert an upper bound on the time to receipt and processing of each message. The exact response time requirement is a function of such factors as the rate with which a process receives messages on a given input channel. Ultimately, these rates are functions of the rates at which data arrives from external sources. These semantics provide a framework both for expressing processor-time-dependent computations and for reasoning about the real-time behavior of programs.

YARTOS itself supports two basic abstractions: *tasks* and *resources*. A task is an independent thread of control (*i.e.*, a sequential program) that is invoked in response to the occurrence of an event. An event is a stimulus that may be generated by processes external to the system or by processes internal to the system. We assume events are generated repeatedly with a (non-zero) lower bound on the duration between consecutive occurrences of the same event. Events are realized through a message abstraction. When an event is generated a message is sent to a task (or tasks) that has registered with the kernel as being interested in the event. Ultimately tasks are reactive in the sense that they execute only in response to the arrival of messages (*i.e.*, events). The processing of each message by a task must be completed before

a well-defined deadline. The invocation intervals and deadlines for a task are derived from constructs in the higher-level programming model.

For many applications, the coupling of timing constraints and message arrivals is unnecessary and often undesirable. YARTOS provides a facility for non-time constrained communication based on shared memory. A resource is a software object (*e.g.*, an abstract data type) that encapsulates shared data and exports a set of procedures for accessing and manipulating the data. Like a monitor, resources guarantee mutually exclusive access to the data they encapsulate. Resources are accessed indirectly through the kernel. Support for resources is included in the kernel to ensure *priority inversions* do not occur — a phenomena in which low priority processes exclude high-priority processes from accessing time-critical data, thus causing the high priority processes to miss deadlines [Sha *et al.* 90].

YARTOS allows applications to specify the real-time rate at which they desire to make progress. A rate is defined as the minimum number of executions of a task that must be completed within a finite length interval of time. For example, a common rate specification is one execution every *p* time units. Such a rate is used for tasks controlling the video digitization hardware (1 execution every 33 ms.). Since tasks are reactive, the specification of a processing rate is, in essence, a specification of the expected rate at which events will be generated. For example, the digitization processor generates 1 event every 33 ms. A more interesting rate specification is used in an application that reads video data from a disk and displays it in real-time on a display. At a high level there is a user task which initiates the display of a frame of video that executes once every 33 ms. (clocked off the the display hardware). The video data that is display is generated by a set of device control tasks that execute *b* times every 33 ms. (clocked off the disk controller) where *b* is the number of disk blocks required to store a single frame. Note that the digitization task in the former example is responding to events in true real-time. That is, frames are being digitized as they are generated. Because of this YARTOS need not (and indeed does not) buffer events for the task. In the case of the disk control task, YARTOS must be prepared to buffer *b* events.

For a given workload (a set of tasks and resources), the goal of YARTOS is to guarantee that (1) all requests of all tasks will complete execution before their deadlines and (2) no shared resource is accessed simultaneously by more than one task. We have developed an optimal (preemptive) algorithm for sequencing such tasks on a single processor [Jeffay 90]. The algorithm is optimal in the sense that it can provide the two guarantees whenever it is possible to do so (*i.e.*, whenever the processor is not overloaded). Moreover, an efficient algorithm has been developed for determining if a workload can be guaranteed a correct execution. This algorithm forms the basis for a resource reservation protocol that is executed prior to the

execution of YARTOS tasks (*e.g.*, prior to the start of a video conference by workstations participating in the conference).

In addition to its academic value, the optimality of the YARTOS resource allocation policies are important for effectively trading-off processing requirements for guaranteed response time. If YARTOS cannot guarantee a correct execution to a process, feedback can be provided on why the guarantee is not possible. The optimality properties ensure that the reasons for the lack of a guarantee are fundamental in nature. A programmer can typically achieve a compromise guarantee either by (1) relaxing the response time requirements, by (2) improving the execution time of one or two specified processes (*e.g.*, by using a different algorithm or requiring less precision in the outputs of an algorithm [Lin *et al*. 87]), or by (3) relaxing the rate at which processes make progress (*e.g.*, by reducing the rate at which video frames are generated). In the latter case, events will be generated that the process will be unable to respond to (the task will not be scheduled in response to events that occur too frequently for its current rate specification). Applications must be written to deal with this situation.

Summary

To be effective and desirable, multimedia applications require real-time computation and communication services from an operating system kernel. The extent to which applications require such services will be a function of the interactive nature of the application and the architecture of the audio and video subsystem. I believe a salient requirement to be that of guaranteed processing rates of data from external devices or peer processes — where the rate is precisely defined by an application program.

The YARTOS kernel supports such a notion of guaranteed processing rates. While in practice this basic mechanism has not seamlessly supported the real-time requirements of multimedia applications such as a workstation-based conferencing system, it has been sufficient to construct the system. In particular, the accuracy of the YARTOS schedulability analysis has been most useful as it has allowed us to concentrate on issues of logical correctness within the application while ignoring altogether efficiency/performance considerations. The desired processing rate of each task is made known to the kernel and the kernel provides a guaranteed response time to each task that is sufficient for ensuring the required processing rate is achieved.

References

Anderson, D.P., Tzou, S.-Y., Wahbe, R., Govindan, R., Andrews, M., 1990. *Support for Continuous Media in the DASH System*, Proc. Tenth Intl. Conf. on Distributed Computing Systems, Paris, France, (May), pp. 54-61.

Bosco, P., 1991. *RS/6000 Workstation Serial Interface: Packet video testbed overview*, Proc. MCNC/NSF Packet Video Videoconference Workshop, Research Triangle Park, NC, (December).

Govindan, R., Anderson, D.P., 1991. *Scheduling and IPC Mechanisms for Continuous Media*, Proc. ACM Symp. on Operating Systems Principles, ACM Operating Systems Review, Vol. 25, No. 5, (October), pp. 68-80.

Harney, K., Keith, M., Lavelle, G., Ryan, L.D., Stark, D.J., 1991. *The i750 Video Processor: A Total Multimedia Solution*, Comm. of the ACM, Vol. 34, No. 4 (April), pp. 64-79.

Hopper, A., 1990. *Pandora — An Experimental System For Multimedia Application*, ACM Operating Systems Review, Vol. 24, No. 2, (April), pp. 19-34.

Jeffay, K., 1989. *The Real-Time Producer/Consumer Paradigm: Towards Verifiable Real-Time Computations*, Ph.D. Thesis, University of Washington, Department of Computer Science, Technical Report #89-09-15.

Jeffay, K., Stone, D., Poirier, D., 1991(a). *YARTOS: Kernel support for efficient, predictable real-time systems*, in "Real-Time Programming," W.A. Halang and K. Ramamritham (eds.), Pergamon Press, 1992, pp. 7-12.

Jeffay, K., Stone, D.L., Smith, F.D., 1992(b). *Kernel Support for Live Digital Audio and Video*, *Computer Communications* (to appear). (Also in Proc. Intl. Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991, Springer-Verlag.)

Lin, K.-J., Natarajan, S., Liu, J.W.-S., 1987. *Imprecise Results: Utilizing Partial Computations in Real-Time Systems*, Proc. of the Eighth IEEE Real-Time Systems Symp., San Jose, CA, (December) , pp. 210-217.

Luther, A.C., 1990. "Digital Video in the PC Environment," McGraw-Hill, Second Ed.

Rangan, P.V., Vin, H.M., 1991. *Designing File Systems for Digital Video and Audio*, Proc. ACM Symp. on Operating Systems Principles, ACM Operating Systems Review, Vol. 25, No. 5, (October), pp. 81-94.

Sha, L., Rajkumar, R., Lehoczky, J.P., 1990. *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*, IEEE Trans. on Computers, Vol. 39, No. 9, (September), pp. 1175-1185.

Terry, D.B., Swinehart, D.C., 1988. *Managing Stored Voice in the Etherphone System*, ACM Trans. on Computer Systems, Vol. 6, No. 1, (February), pp. 3-27.

Wirth, N., 1977. *Toward a discipline of real-time programming*, Comm. of the ACM, Vol. 20, No. 8 (August), 577-583.