# Operating System Issues for Continuous Media

Henning Schulzrinne

May 15, 1995

## 1 Introduction

Operating systems play a crucial role in supporting multimedia applications. In this chapter, we will investigate the problems that are introduced by multimedia and some approaches to their solution. General introductions to operating systems can be found in [1,2], with specific surveys of the Unix operating system in [3,4]. Although only the combination of network and operating system design can guarantee performance, network issues seem to have required far more attention in the past. As higher-speed LANs touting integrated services are becoming available, operating system deficiencies are becoming more apparent. These and some attempted approaches are the subject of this chapter, with a strong emphasis on implementation-related work rather than theoretical results from scheduling theory.

First, it is helpful to distinguish between *continuous media* [5] and *multimedia*, where the latter includes the former. Continuous media are characterized by a timing relationship between source and sink, that is, the sink has to reproduce the timing relationship that existed at the source. The most common examples of continuous media include audio and motion video, but MIDI commands also belong in this category [6]. Continuous media can be *real-time (interactive)*, where there is a "tight" timing relationship between source and sink, or *playback*, where the relationship is less strict[1]. Video conferencing is the stereotypical application for interactive continuous media. For interactive continuous media, the acceptable delay is often governed by the human reaction time, or issues of echo for audio. These delays are on the order of no more than a few hundred milliseconds. The tightest delay constraints are imposed by bidirectional audio. If (currently rather expensive) hardware echo cancellation is to be avoided, delays of no more than around 40 ms are tolerable. For playback applications, the delay is limited mostly by the amount of buffering that can be supported. In addition, for some playback applications, the total delay, including network delays, is limited by the desired reaction time for control commands. For example, a video-on-demand playback application should be able to react to a "reverse" command within no more than about half a second.

In the operating system context, multimedia indicates the integration of different media within the same system, some of which may well be continuous media of both playback and interactive type, but also can be still images, text or graphics. One specific problem, covered in chapter XXX, is the coordination in time (synchronization) of these media.

[1] For simplicity, we also imply recording as for a tape recorder when using the term "playback".

Because its characteristics differ most markedly from traditional data types supported within operating systems, this chapter will focus largely on *continuous media*. While it is often assumed that continuous media is challenging because it imposes the highest data rates or processing on the system. This may not necessarily be so; graphics or transaction processing may well impose larger loads on a larger range of system components, but the timing requirements and the guaranteed throughput required for continuous media force specific consideration in designing operating systems.

The design of multimedia operating systems is fundamentally determined by how the media data is handled within the computer system. We can distinguish a control-only approach from a fully integrated approach. In a control-only approach, the continuous media data does not touch the operating system or main memory, but rather uses a separate infrastructure. For the case of video, the control-only approach simply connects analog video to a separate monitor or uses analog picture-in-picture techniques. The earliest experiments in integrating analog video such as [7] had the workstation control a VCR or laser disk player, with video either displayed on a separate monitor or fed through an analog mixer. For monochrome, a simple control plane can be used, while for color, chromakeying is a traditional technique long used for TV production. Analog mixing makes it difficult to integrate video into, say, the windowing system, so that the user perceives it as an attachment rather than integral part of the system. There is also likely to be a rather low upper bound on the number of concurrent video windows. Audio control functionality like talker indication or on-screen VU display cannot be readily supported. While this architecture imposes some mild control timing constraints on the operating system, it is not likely to require fundamental changes. The Etherphone [8], VOX [9], DVI [10] and IMAL [11] systems are examples of the separation of control and data for audio.

Instead of analog mixing, the Pandora system [12] integrated video in digital form, but through a separate video processor and a pixel switch that decided for each pixel whether to display the workstation graphics stream or the external live video source. This allows mostly seamless integration into the windowing system, but with a separate data path and the inability, for example, to capture screen dumps of live video. The combined multimedia workstation also requires two separate network interfaces. A more integrated approach has live video and workstation graphics share the same frame buffer, as is done for many workstation-based video boards [13].

A more recent, fully digital version of separate or separable data paths for continuous media is found in the idea of desk-area networks [14–17] that extend cell-based communication into the workstation, with a cell switch replacing a traditional bus or the path through memory. However, this approach offers the choice of either treating continuous media as data to be processed or simply switched from, say, network device to display device. If data does not touch the CPU on its path through the system, operating system requirements are much relaxed. Thus, for our purposes, we will not discuss this mode further.

Given that many continuous media applications can be realized without passing media data through the CPU, using one of the approaches outlined, the reader may ask why integration is desirable at all, given the complexity.

Naturally, "AI-like" functions like image categorization will require CPU intervention, but other, lower-level operations like like encoding and decoding, scaling or sampling rate conversion can be done either in hardware or software. Software generally has a price advantage. In particular, the incremental cost of adding video to an existing workstation can be close to zero if only decoding is desired. Except for switch-based backplane architectures, there is also another pragmatic reason to involve the CPU in that most workstation architectures are simply not designed to allow adapter-to-adapter communication. With evolving standards in the area of network protocols and media compression, software-intensive approaches offer far more flexibility. The type of advanced media functionality described in [18] is likely difficult to achieve when continuous media bypasses operating system and CPU.

There are other advantages as well, in particular scalable performance. Software decoding, for example, can display several windows of live video simultaneously (depending on workstation performance), while most hardware decoders only support a single output window. Examples include the decoding of video [19, 20], both encoding and decoding of video [21–23] and audio [24], as well as the processing of MIDI data [6]. At least as important as the necessary processing power is the integration of standard continuous-media I/O devices on the motherboard, currently, devices for high-quality audio, but probably soon video as well. The integration of these devices in workstations has had the salutary effect of accelerating integration of continuous-media support in the vendor-supplied operating systems[2].

While the increasing computational power of workstations should make possible fairly extensive manipulation of continuous media, a number of factors reduce the ability of workstations to deal with continuous media below what might be expected from pure MIPS or SPECmark benchmark numbers. Continuous media stresses a number of architectural areas poorly covered by traditional CPU-oriented benchmarks and areas where performance has not increased with raw CPU MIPS performance, in particular interrupt latencies, context switching overhead and data movement. Audio, for example, is often handled in small chunks to reduce packetization delays. (A 20 ms audio packet sampled at 8 kHz and coded as $\mu$-law consists of 160 bytes.) These small packets cause a large interrupt load for network devices and frequent context switches for applications. (In the example, potentially 100 context switches per second.) MIDI has a low data rate of 31 kb/s, but extremely tight latency requirements of around 1 ms.

Other continuous media types produce large data rates. CD-quality audio, for example, generates samples at rates of around 1.5 Mb/s. Video exacerbates the demands by requiring data rates of about 1.2 Mb/s for MPEG-compressed, VHS-quality video [25], 20 Mb/s for compressed HDTV, up to 200 Mb/s for 24-bit (RGB) uncompressed video at 30 frames/s (640 by 480 pixels) or 1.2 Gb/s for uncompressed HDTV. While networks and file servers will rarely carry uncompressed video, operating systems may have to handle these if decompression is done in software.

Unfortunately, as observed by Ousterhout [26], both memory bandwidth and operating system functions such as context switches or interrupt handling have not improved at the same rate as CPU instruction cycles per second. Even though basic DRAM memory speeds have not increased much in the last decade, workstation designers have been able to improve benchmarked performance by enlarging and speeding up static-RAM-based cache memory. Cache helps with some continuous media tasks, such as software compression and decompression, but the large amounts of data flowing through the CPU and memory system will likely decrease overall cache hit ratios, with system performance dominated by slower main memory and I/O accesses. This memory bottleneck was observed, for example, for a software MPEG decoder [19], where memory bandwidth, not the processing for the inverse cosine transform limited the achievable frame rate. Disk access speeds also

[2]It should be noted, however, that many 'multimedia PCs', despite featuring a CD-quality audio interface, are unsuitable for *interactive* continuous media because the cannot handle concurrent reading and writing of data, i.e., they are half duplex. Also, their DMA mechanism is designed for playback of sounds, not low-latency interaction.

have not improved dramatically. Disk caches are of limited use since most continuous media is accessed sequentially and only once, with the exception of large video-on-demand servers.

In general, the relationship of system resources to requirements can be depicted in a graph like Fig. 1. In the upper left region, resources are insufficient even for a single stream, with no other system activity. As we move into the region of scarce resources, a limited number of continuous media streams can be handled, as long as they and any non-real-time processes are controlled carefully, i.e., appropriately scheduled and possibly denied concurrent access. As we move further to the right, labeled abundance of resources, we may have enough resources on average for all comers, but may still have to do localized control to avoid missing deadlines for individual streams. Note that this division finds its correspondence in communication resources. There, the window of scarcity reflects the region where admission control is needed.

The data points shown naturally reflect only "typical" usage. In particular, mixing of several audio or video streams, their manipulation (special effects) or higher tempo-spatial resolution can extend the window of scarcity almost indefinitely.

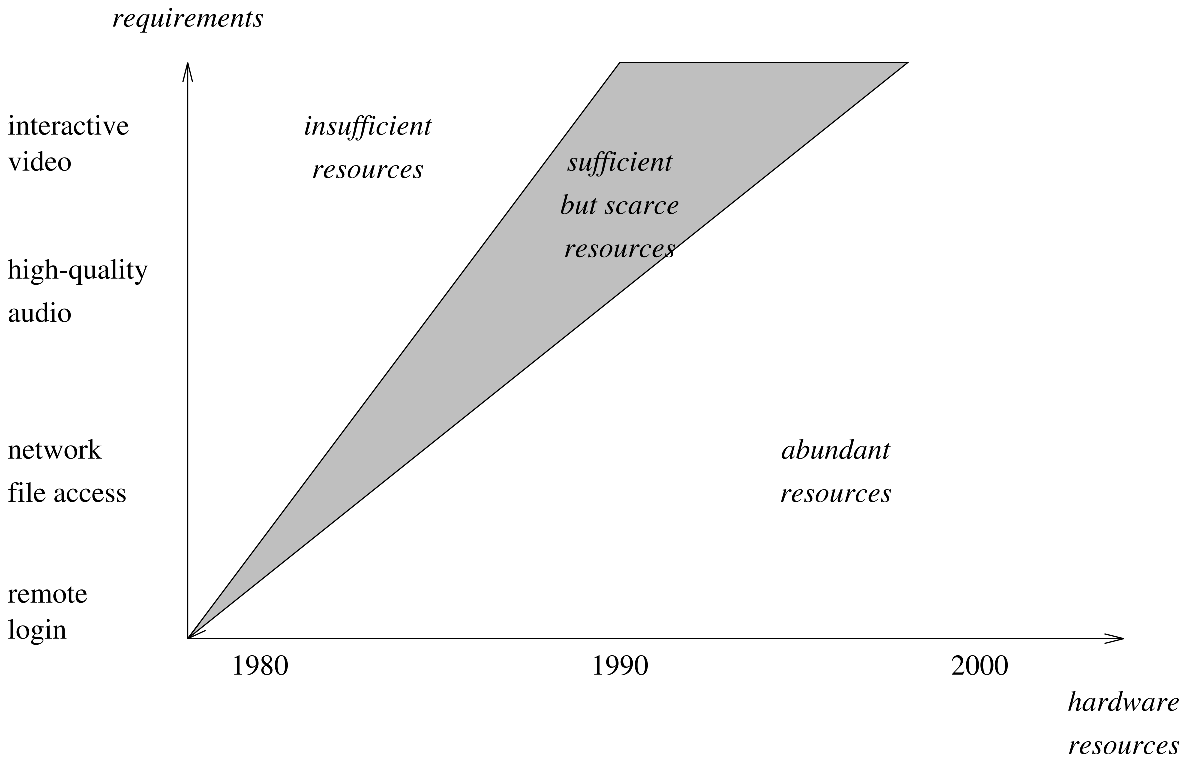


Figure 1: The window of scarcity [27]

Up to now, we have indicated some of the challenges faced by system designers when integrating continuous media into a general-purpose operating system. However, the basic functionality required of a general-purpose, continuous-media enabled operating system remains similar: to arbitrate resource demands, protect processes against each other, and provide abstractions of low-level physical devices. The protection of processes, however, now encompasses not just protection against external memory accesses but also of their negotiated slice of CPU time, I/O bandwidth and other resources. In addition to these "kernel" functions, such enhanced operating systems may also support multimedia applications by offering security measures, protocols for data sharing and orchestrating multimedia data.

In the following sections, we will discuss two of the major bottleneck system resources, CPU and disk, and then look at higher-level system support through libraries, security issues and finally discuss some systems explicitly designed for continuous media applications. We will discover that many of the resource issues faced by the operating system have counterparts within integrated services networks, while the possible design approaches are somewhat more flexible due principally to the reduced latencies within an operating system compared to a distributed network. Also, an operating system can usually obtain a global view of the system, unlike a distributed network.

# 2 CPU Scheduling

## 2.1 Characteristics of Continuous Media

The characteristics of multimedia and collaborative data sources on networks and operating systems can be roughly divided according to Fig. 2, adapted from [28]. Traditional (hard) real-time events occur periodically; if an event is not processed or not processed by its deadline, this is considered a catastrophic failure [29], e.g., a chemical plant explodes. Interactive audio and video typically also arrive periodically from input devices and roughly so from the network, but with far less stringent reliability constraints. Other system events, such as mouse movements, keystrokes or window events, have to be processed within a few hundred milliseconds and reliably, but do not have a predictable arrival patterns.

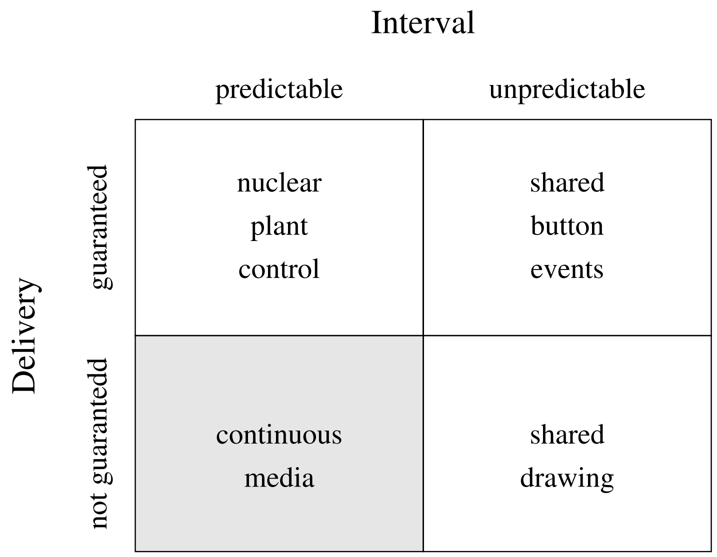


Figure 2: Characteristics of time-critical data [28]

## 2.2 Workload Characterization and Admission Control

Current timesharing operating system do not expect CPU time commitments from processes and in turn cannot make commitments of response times. Process creation requests are only rejected when some other resource, such as process slots or swap space, is insufficient. This behavior is analogous to a datagram network.

Just as networks have difficulties making QOS guarantees unless both the traffic is bounded in burstiness and packet scheduling isolates streams from each other, a process scheduler needs to do the same. Unfortunately, applications generally do not know their resource requirements, except perhaps by historical statistics. The difficulty is even worse than for estimating network requirements (e.g., cell rate statistics), as subtle configuration changes or hardware upgrades can drastically alter the amount of execution time required for the continuous media task. To confound matters further, execution times can depend on the behavior of other processes. Generally, the busier a system is, the more per-process CPU time will be required as CPU and MMU caches get flushed more frequently, there are likely to be more involuntary context switches, run queues to be searched by the scheduler are longer, and so on. This pattern of increased resource consumption just when the system is busy has no analogue in the communication realm, except perhaps for overloaded routers.

Despite these dependencies, it appears that most continuous media processes could provide reasonable upper bounds on their resource usage, at least as long as their repertoire of operations is limited to simple combinations of frame retrieval, compression/decompression, dithering and the like. Attempts at estimation have been made, e.g., by Jones [30] based on prior execution. However, as pointed out by [31], upper or pessimistic bounds may lead to poor resource utilization and unnecessary denials of service. However, they may also provide the necessary head room to non-real-time applications, so that a larger fraction of the system capacity can be reserved for real-time tasks. In CSCW applications, for example, it appears to be common to have a mix of interactive, continuous media applications and more traditional data applications like shared windowing systems, whiteboards, conference control and the like. Another approach, optimistic bounds, reserves only the bare minimum of resources for a real-time task and allows for occasional glitches and service degradations, things we have come to tolerate in analog telephony and over-the-air television.

Interactive continuous-media applications are characterized by a cycle, occurring at frame rate or audio packet rate, consisting of waiting for available data either from the network, an audio input device or framegrabber, followed by CPU and memory intensive processing. While the media input device events occur at regular intervals, network arrivals can be quite bursty, particularly if an application receives packets from a number of sources. For audio and intraframe-coded video, the each such round requires a fairly constant amount of processing, while, for example, the processing and memory accesses required for MPEG I frames are likely to be much larger than for B and P frames, with strong dependency on the image content. Thus, audio input follows a traditional real-time model of periodic, predictable needs for CPU time, while all other tasks have either unpredictable arrival instances or widely varying processing requirements. The application is likely to have no notion how much processing is required for each set of images and the particular processor architecture it happens to be running on.

This periodic resource requirement is actually not too far removed from the characteristic pattern of hard real-time applications, as they might be found on a factory floor, sampling sensors at a fixed rate and computing control responses. Playback applications tend to be much burstier, in that they may read a number of frames from disk, decompress until the process is preempted and continue.

One simple model for continuous media requirements is the linear bounded arrival process (LBAP), originally developed by Cruz [32,33] and suggested by Anderson [5,27,34,35] as a model for both communication networks and operating systems. In the LBAP, the message arrival rate in any interval $t$ may not exceed $B + Rt$, where $B$ is the maximum burst size and $R$ is the average arrival rate. Instead of messages, other units like required CPU seconds or number of bytes to be read from disk could be used.

For scheduling, we can distinguish between *critical* and *workahead* processes. First, we define the logical arrival time of a message as the time it would have arrived had the generating process strictly obeyed its maximum message rate. More formally, the logical arrival time $l(m_i)$ is defined as

$$l(m_i) = a_i + \frac{w(m_i)}{R},$$

where $a_i$ is the arrival time. The workahead $w(m_i)$ is defined as $\max(0, w(m_{i-1}) - R(a_i - a_{i-1}) + 1) = \max_{t_0 < a_i}(0, N(t_0, a_i) - R|a_i - t_0|)$. The workahead (or backlog) measures how much the process is "ahead of schedule" relative to its long-term rate $R$. Note that with this measure, a single late-arriving message can delay the logical arrival time of all subsequent messages and thus their processing. Also, the burst size $B$ does not enter the computation of the logical arrival time. A message after its logical arrival time is termed *critical*, while one before is termed *workahead*. An example is shown in Fig. 3.

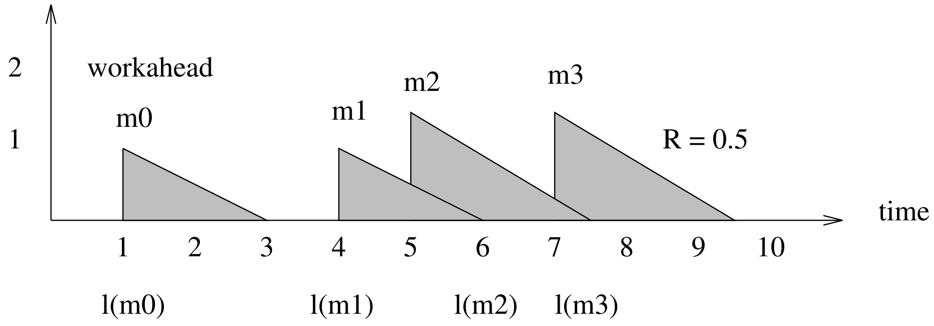


Figure 3: Example of workahead function [5]

## 2.3 Existing Systems

Before addressing the issue of CPU scheduling and (implicitly) resource reservation, it helps to establish that current practice in general-purpose *and* real-time operating systems is insufficient for the desired quality of delivery of continuous media. On the positive side, most current Unix-based workstations can keep up with a single audio or video stream as long as there are no other applications and the continuous media stream is only piped through the system with moderate processing[3]. However, even for a quiescent system, there can be significant variation in the interdeparture and application-visible interarrival times caused by system demons or process rescheduling, as measured in [36].

[3] This accounts for the success of multimedia demos at trade shows.

The reason for these variations are found in the properties kernels and process schedulers Both System-V [4] and BSD-derived schedulers [3, p. 86] use a multilevel feedback queue, where processes are served round-robin within a priority level, but moved from one level to another based on CPU time accumulated and a preference for I/O intensive processes. The priority level is adjusted only if a process exhausts its CPU time slice of (typically) 100 ms or if it has been sleeping longer than a second. Thus, typical continuous media applications should maintain their priority throughout, but other processes could preempt it when starting[4] or if they can "convince" the kernel that they are more I/O intensive. A process is only preempted after using its timeslice or when returning from a system call. Thus, even after raising the priority of the continuous-media process, it can suffer delays of up to 100 ms until a lower-priority process that got scheduled during the wait period resurfaces from the kernel or exhausts its time slice. Residence time in the kernel is not bounded at all, regardless of the priority of the process. The problems with this time-sharing scheduling discipline have been empirically evaluated by Nieh *et al.* [37], where a video application, a typing simulator, the X server and a compiler-like compute job that continuously spawns new short-lived processes compete for resources. Simply adjusting priorities (known as "nice" values in Unix) may help, but often has counterintuitive effects, in that *lowering* the process' priority may, say, increase frame rates.

As noted, Unix process scheduling decreases the process priority with accumulated CPU usage. However, for periodic processes, sampling aliasing can dramatically skew the estimate so that each video or audio frame gets charged the full CPU tick or nothing at all [38], leading to periodic oscillations in process priority as application intervals and measurement intervals drift with respect to one another. Random sampling intervals solve this particular problem.

Real-time Unix systems [39, 40] generally implement a fixed-priority, preemptive schedulers and are thus not suited for a general-purpose system shared by realtime (continuous-media), interactive and batch applications.

## 2.4 Priority for Real-Time Processes

There are a number of conceptually simple fixes that can be applied to standard operating systems like Unix to improve their behavior for continuous media applications without giving up on a general-purpose operating system.

One problem noted in the preceding section was the unbound kernel residence time. Newer versions of Unix SVR4 [41] and the Mach operating system implement a threads-based, fully-preemptible kernel, so that low-priority processes executing kernel code cannot prevent higher-priority processes from running. This change reduced dispatch latencies from 100 ms to about 2 ms on a standard workstation. Another approach to reducing maximum kernel residence time is by introducing preemption points [42], where system routines such as fork, exec or exit check whether any high-priority user processes are ready to run. It should be noted that the interruptability of kernel code comes at a price in overall performance, as threads have to execute semaphores and the run queue has to be checked at every preemption point.

Unix SVR4 also introduced installable scheduling classes [6, 41], including a real-time class with fixed priorities (which are always higher than those processes within the timesharing class). Processes in the real-time class can either execute until blocked or finished, or a time limit can be

[4] Process priority is highest at the time the process is created, unless manually changed by the superuser.

associated with the process. Since the priority of the real-time class has to be higher than that for system processes like the swapper in order to guarantee delay bounds, use of this class can obviously have dire consequences for system behavior[5]. This is documented in [37] through the example of a video display process that can cause the whole system to become unresponsive. However, the example chosen in [37] is clearly inappropriate for the real-time class as it is a CPU-bound process that will dominate the CPU if given a chance. A high-priority real-time class can only work if the processes in that class relinquish the processor so that lower-priority processes are not starved. Nieh *et al.* [37] present a new timesharing class that provides a better balance between interactive, real-time and batch applications.

It should be noted that running a continuous-media application within an elevated-priority real-time scheduling class by itself may not be sufficient if this application has to rely on other system services, which are likely to execute as timesharing or system processes. Examples include the X window server for video output and the streams handler for audio input and output. To avoid wholesale priority elevation of all system services, system services have to be threaded, with only those threads operating on behalf of real-time applications getting improved service. Indeed, even within a real-time application, there may be significant non-real-time components, particularly the user interface or logging. Again, only separate threads of control, with separate system-level scheduling can avoid these problems. Threads within most threads libraries, however, are only scheduled within the time-slice of the overall process.

## 2.5 Priority Inversion

*Simple priority inversion* occurs if a high-priority process has to wait for a resource held by a low-priority process. *Unbounded priority inversion* occurs if this low-priority process, while holding the resource, is preempted by a medium-priority process, delaying release of the resource and blocking the high-priority process for an indeterminate amount of time [43]. There are a number of remedies [44], the simplest of which is the basic priority inheritance protocol. The basic priority inheritance protocol attempts to bound the duration of priority inversion during resource blocking by having the high priority thread propagate its priority to all lower priority threads that block it [41].

## 2.6 Deadline-Based Scheduling

If several concurrent continuous media streams are to be handled by the operating system, higher priorities need to be assigned to processes with shorter deadlines. However, this preemptive priority may lead to processes with lower priorities missing their deadlines, even though the system could have scheduled both high and low priority processes on time.

To avoid this difficulty and the problem of having to manually assign priorities, we can schedule processes earliest deadline first (EDF) [45]. EDF does not require processes to be periodic; it also has nice optimality properties in that it yields a schedule that avoids missed deadlines if that is possible at all. EDF is also used in communication networks, but for CPU scheduling, the preemptive variety is appropriate, with the readiness of a new process with a closer deadline preempting the currently executing process. EDF scheduling within the Unix operating system has been implemented by Hagsand and Sjödin [42], for Mach by Tokuda *et al.* [40] and Nakajima *et al.* [46].

[5] Not surprisingly, use of this class requires superuser privileges.

In [42], a process starts towards its deadline if one of the designed file descriptors become ready, triggered, for example, by the arrival of a video frame from a framegrabber or the network interface. Real-time processes are designated by waiting on a file descriptor passed to a variation of the standard `select` system call. There is no schedulability test, however, so real-time processes are not protected against each other.

The YARTOS (Yet Another Real-Time Operating System) [47,48] OS uses a related approach. Here, the deadline clock starts to tick with the arrival of messages (events), a generalization of the file-descriptor approach mentioned above. Events have associated deadlines and a defined minimum interarrival time. With this and the specification of resources needed and available, the kernel can offer guarantees to tasks.

Another approach to deadline-based scheduling is the deadline-workahead scheduler (DWS) [35]. Using the LBAP workload characterization described in Section 2.2, a simple policy can be formulated: critical processes are scheduled EDF, regular interactive (non-real-time) processes have priority over workahead processes, but are preempted when those processes become critical. Finally, the scheme defines background processes as those executing when the system would otherwise be idle. The approach is thus seen as a refinement of the pure EDF, with a built-in bound on the resource consumption of individual message streams.

Either priority classes or deadline scheduling offer a tempting target for application writers to "improve" the performance of their application while penalizing other users or applications, leading, in the worst case, to an "arms race" between applications and defeating kernel scheduling altogether [49]. Making class assignments accessible to the super user only on a per-process basis makes them basically unusable for normal user applications.

## 2.7 Imprecise Computation and Adaptive Applications

Some CPU-intensive continuous media tasks are characterized by the fact that they can be carried to different degrees of precision. As an example, consider a video encoder where the motion estimation search can be more or less exhaustive. If a process runs out of time, it can terminate before completion with a useful, but less precise result [50]. The system could signal to the process that "times's up", causing it to terminate that round of processing and avoiding missing its own deadline or endangering that of another message. These type of tasks are labeled as *imprecise computation* or IRIS (increasing reward with increasing service). In principle, imprecise computation offers a graceful degradation of the media quality within a continuous media system as system load increases.

This seemingly straightforward and appealing approach has a practical problem in that the typical continuous-media computation consists of a CPU-intensive task that can be varied in length, followed by "packaging" (e.g., Huffman coding for JPEG, or generating the MPEG frames) plus any system overhead needed to render or transfer the processed data. While there are theoretical investigations in the properties of inprecise computation (e.g., [51,52]), there does not seem to be any practical experience with actually using such algorithms for codecs.

A slightly different adaptation algorithm works on a longer timescale. Instead of rejecting new real-time tasks, the system detects overload and then renegotiates resource requirements with these tasks [53].

Unlike most batch applications that are useful over a range of execution durations of a hundred or more, most continuous media applications have a fairly narrow range of adaptability. Video frame rates can be varied by maybe a factor of 2 (from 30 to 15 frames a second, say) without being fundamentally altered in usability (from moving video to changing still images). It should also be noted that adaptation of continuous media for the network could either cooperate with or defeat operating system-induced adjustments. Frame rate reductions would relieve load both on the network and the end system, while changing to a lower-bitrate coding would likely be more computationally intensive. Thus, network-motivated adjustments have to be carefully coordinated with local system resources.

```
*** Scheduling
Ande9308:Metascheduling
Toku9311:Dynamic
Auds90:Real
Govi9103:Scheduling
Merc9405:Processor
Auds9105:Hard
```

# 3 Memory Allocation, Paging and Swapping

In the past, increased swapping and paging was a concern for system performance, but it appears that increased memory has largely eliminated swapping, except for dormant applications. In any event, given the ever larger ratio of CPU speeds to disk speeds, reasonable system performance cannot be obtained if processes have to retrieve memory pages after initial load during startup.

However, with increasing availability on workstations, demands on memory have grown, from caches for disks and network information retrieval systems, to temporary file systems to speed up compilations [49]. Instead of a relatively small number of active processes when user interactin was through terminals, each user now typically has sets of mostly inactive applications. Typical per-user process counts are probably fairly close to a busy timesharing system of some years ago. Any of these processes, when awakened, will grab pages from other processes, including those with real-time constraints. Least-recently-used (LRU) policies offer some protection to periodically executing processes, but there are likely parts of an interactive application that are executed only infrequently, e.g., to update statistics. For single-threaded applications, paging in these parts of the continuous media application could then delay all real-time handling within the same application. Memory access is typically not tied to scheduling priority so that a low-priority batch job can easily interfere with a high-priority continuous-media task.

Simple tools like indicating to the linker to colocate certain memory segments could lead to decreased working sets [49], but is generally not supported.

# 4 Performance Enhancements

In this section, we investigate approaches to streamline processing for high-data rate, low latency applications. We also mention special requirements on common functionality found within current operating systems, e.g., high-resolution clocks and clock synchronization.

## 4.1 Clocks and Timers

High-resolution clocks and timers are required for low latency, event-driven applications. Typical Unix clocks have a resolution of only 10 ms or coarser, while resolutions of below a millisecond are probably required for applications like MIDI. Timers may have large cumulative errors, that is, if a timer is rescheduled periodically, the total accumulated time may be significantly larger than the sum of the timer values. For intermedia synchronization, it is important that the system clock can be read with low overhead and without the possibility that the process will be preempted at the end of the system call.

Generally, the system timer is ill suited to clock the gathering of continuous-media data, as it likely runs at a slightly different frequency than the sampling clock on the audio/video data acquisition hardware, leading to buffer overflows or starvation both for input and output. It appears best to use the hardware to clock both data acquisition and playback.

Since the notion of time is central for continuous media, operating systems for multimedia should also provide basic support for synchronization of media streams. Synchronization between media streams has been the topic of a large body of research [54–57]. It appears, however, that once synchronized clocks are available with clock differences of a few milliseconds [58], the problem is largely solved. Explicit synchronization algorithms appear to provide special-case multi-party clock synchronization. Some further experimental work on playout synchronization in different networks is useful, although the need for sophisticated algorithms arises primarily in lower speed packet networks like the Internet [59].

## 4.2 Interrupt Handling

As mentioned in Section 1, some continuous media types may cause high system load not due to their bandwidth but rather their high interrupt and context switching frequency. It has been observed elsewhere that both have come to dominate actual protocol processing, particularly if they cause caches to be flushed and MMU tables to be invalidated. Because of their relatively low interrupt and context switch overhead, operating systems that do not offer preemptive multitasking and separate address spaces (e.g., Microsoft Windows 3.1 or Apple System 7) may have significantly better real-time performance than UNIX derivatives. Because of the lack of separation of a kernel and user space, processing is often done within interrupt handlers, offering low latency, but clearly at the cost of higher, non-predictable delays for other processes [6]. One approach for System-V-derived kernels may be the use of modules pushed onto the streams processing stack [60], either as pseudo-code or as compiled modules. However, like upcalls from kernel to user space, they pose protection problems and may contravene attempts to limit kernel residence times. The Berkeley packet filter [61] provides one example (in the somewhat different domain of selecting and processing network packets for debugging) of how to construct a simple, restricted interpreted language which is downloaded into the kernel and then executed for every arriving packet.

In a BSD-derived operating system, interrupts triggered by hardware, so-called "hard interrupts", preempt any other system activity (except other interrupts of higher priority). Therefore, handlers for these interrupts commonly attend to only the hardware-related tasks, leaving the actual work to be done (e.g., the copying of data) to a so-called "soft" interrupt, of lower priority than all other interrupt handlers, but still preempting other user processes. Some such interrupt handling (e.g., for background network traffic) is typically of lower priority than real-time processes and, given sufficient background load, it may pay to defer those [42]. However, deciding whether, for example, a packet belongs to a network association deserving high-priority treatment or not may not be trivial at hard interrupt time and could outweigh any gains. As with assigning processes to scheduling classes, what looks like a low-priority packet arrival may actually be in the critical path for a real-time application.

### 4.3 Memory Access and Copying

As mentioned earlier, continuous media applications often need to move data continuously and at high rates from one I/O device to another, e.g., from framegrabber to network interface. In a standard Unix kernel with DMA-capable devices, the data is copied three to four times: via DMA to a device-driver managed buffer in RAM, then by the CPU to internal data structures like *mbufs*, back into the output device's DMA buffer, and finally via DMA into the output device [62]. If no further processing is required, direct driver-to-driver data transfers could off-load the CPU, although the competition for bus cycles may still reduce system performance unless the peripherals can use a dedicated I/O bus as in the microchannel architecture.

Another approach, used also by the Mach operating system, is to avoid copying by remapping pages from one process to another. However, the attendant overhead in updating kernel tables and remapping pointers may approach the cost of copying for all but the largest messages. Anderson [5] introduces a restricted remapping in the context of the DASH message-based operating system for continuous media, where only a special memory region can be remapped and only between the same virtual addresses in sender and receiver address spaces.

## 5 Network Interface and Protocol Processing

packet voice: interrupt load; copying for large messages, fixed overhead for small messages (ToN paper)

[63] [64] (ATM)

## 6 Libraries, Toolkits and Application Environments

While the libraries described in this section are typically implemented in user space rather than the operating system kernel, they provide the necessary isolation of applications from details of the hardware and from the behavior of other processes. In general, the audio and video extensions, servers and libraries would also be natural places to locate admission control.

### 6.1 Audio

While POSIX and X11 allow reasonably quick cross-platform development on Unix systems for many interactive applications, every vendor seems to have their own audio and video programming interface. Generally, both audio and video are mostly sequential devices, so that the basic Unix device or streams model is appropriate, enhanced with the necessary control functionality for adjusting parameters such as sampling rate and volume for audio and brightness and frame rate for video. This device model makes it relatively straightforward to incorporate reading from an audio device and reading an audio clip stored in a disk file.

SGI Irix extends the device model by offering up to four logically independent audio streams, with mixing performed by the audio hardware. Solaris has a single, non-shareable audio device, but allows several processes to control audio parameters that do not affect the encoding of the audio stream. Audio device abstractions should provide for variable triggering depths on input, i.e., the amount of data resident in the low-level FIFOs before the kernel signals to the application that a new chunk of audio data is ready for reading. so that interactive applications can trade low delay for occasional audio glitches, while playback and recording applications can incur a higher delay and fewer context switches. Also, it is currently difficult to ascertain with any accuracy when a particular audio sample was acquired (for audio input) and when it is going to be played (for audio output). Both of these times are important for synchronizing streams and echo cancellation.

The principal deficiency of these audio-as-device models is the lack of resource sharing, i.e., only a single application has access to the audio device until this application explicitly relinquishes control (a kind of cooperative multitasking). System sounds and other background sounds [65] are difficult to implement this way.

An audio API (or engine) should allow sharing of the single speaker by several applications by either mixing at different volumes or priority preemption. External applications like VU meters [66], recorders or automatic gain control should be attachable to the audio input and output, without having to be replicated for every application. The audio engine should transparently translate encoding, sampling rates and channel counts to the desired output format. It may provide for synchronization between several audio streams or, if possible, between an audio stream and a video or MIDI stream. Separating audio source and sink by a network requires sophisticated playout adjustment at the receiver, particularly if the solution is to be usable beyond an uncongested local area network. Thus, a general system solution seems preferable, rather than every application having to develop its own solution. Design complexities remain; for example, indicating the current talker is more complicated for a system library, as is the compensation for losses or other interventions by the application. Thus, either the audio library has to do perform almost all desirable audio services or very few beyond mixing, volume control and the like.

Existing implementations such as [36] favor the client-server model, following the approach taken by the X window system. Applications contain a clientside library, which transfers audio data to a server running at the workstation with the physical audio input and output devices. The server integrates output requests onto a single physical device either by alternating between them according to some priority order or mixing them in some user-determined weighting.

The Alib client-server library [67] allows to tie together components like telephones, audio input and output devices, speech synthesizers and recognizers, and recorders and playback devices through logical “wires”. Commands, sent through a separate command pipeline, are automatically sequentialized, so that a playback operation can be immediately followed by recording audio.

Some attempts have been made at cross-platform APIs with enhanced functionality [68], but these appear to have a number of shortcomings. Most audio APIs seem designed mainly for playing back and recording audio clips rather than real-time use. Netaudio [69] and AudioFile [70, 71] also abandon the Unix device-as-file model, requiring separate hooks into event handlers, separate read/write routines, and so on. This appears to be a step backwards and makes it difficult to use the same code to read from an interactive audio device and a file containing audio data.

Unlike windowing systems, these libraries have a separate control and data stream. This makes it difficult to change audio parameters at precisely defined points, e.g., to change the audio encoding when switching between two streams. With the current control structure, this is inherently difficult since device control is done asynchronously, causing data in the internal operating systems buffers to be misinterpreted. Some kind of marking of the audio stream is required so that the physical device can be switched at the appropriate instant.

From these discussions, we can identify the following requirements for future work:

- The abstraction should be suitable for high-packet-rate, low-latency interactive use as well as playback and recording.
- It should support a variety of encodings transparently, with in-line switching between encodings.
- Client and server should be able to be located either within the same LAN, with low latency and delay jitter, as well as spread across a WAN, with large delay jitter and packet loss. In other words, the transport mechanism cannot be a traditional data transport protocol; adaptive playout delay compensation needs to be handled.
- Streams should be able to share physical audio devices, either by preemption, mixing or a combination (lowering volume of other streams).
- Server should cache sound clips for repeated play.
- Resource reservation may be needed to ensure that decoding and mixing does not exceed available system capacity.

An operating system enhancement fulfilling these requirements makes it possible to add interactive and playback audio to applications quickly, without having to reimplement relatively complicated media handling in every application.

## 6.2 Video

There seem to be relatively few windowing system extensions that directly integrate motion video. Schnorf [72] describes an extension of the ET++ windowing toolkit where motion video, generated by a hardware or software decoder, becomes a first-class object, being able to be clipped and moved like other windowing objects. The XIL [73] toolkit attempts to shield the user from the complexities of video manipulation, frame grabbing, scaling and compression. In particular, the library automatically composes several operations into a single "molecule", linking, say, frame grabbing and scaling and avoiding unnecessary data copying and normalization. The library automatically downloads the necessary code into the video processor chip on the framegrabber board. However, because of restrictions in what operations can be combined, it is easy to generate sequences where every other frame is scaled and then dropped. Operations that would appear to be associative are not.

### 6.3 Multimedia Control and Scripting

Apple QuickTime and the MuX multimedia I/O server [74] are examples of operating system extensions that attempt to integrate multimedia objects. QuickTime is limited to a single host, while MuX has a similar architecture as the X window system, with clients and servers potentially distributed among hosts. Both are following a video production model, with the ability to compose multimedia objects from tracks of audio and video. Lacking are the ability to integrate real-time video and sophisticated user interaction such as hypertext-like linking, as well the integration of graphics.

Another form of simple integration is offered by Microsoft Windows: system events like executing a file can be associated with multimedia objects.

## 7 Security

Continuous media data poses particular security concerns. For most data, the concern was whether an unauthorized user could read or modify a user's file or process space. If multimedia devices are attached to workstations, the workstation can become a remote-controlled monitoring device [28]. For example, some workstations were delivered with the audio input device set world-readable, allowing remote listening to whoever attached a process to that device [75, 76]. Some operating systems have added the facility that ownership of these devices follows that of the console, which avoids monitoring of the one sitting in front of the workstation by remote parties, but does not solve the problem of superuser access or "forgetting" to shut off video and audio applications when leaving a group office. Tang [77] suggests enforcing mutual viewing, so that if A can see B, B can also see A, but admits that this is hardly practical in a distributed system. In general, use of receiver-oriented multicast for efficiency will make encryption the only viable means of protecting privacy. Zimmermann [75] suggests a capability-based protection system for multimedia operating systems.

If continuous media data is encrypted and decrypted within the application, traditional user-kernel boundaries and attendant copying across that boundary are no longer needed. Buffers can be mapped directly into a number of user memory spaces, shared between user processes. Also, for compressed video it may be possible to encrypt only parts of the frame or of a frame sequence (I frames for MPEG) to save processing time, as long as the remainder of the frame or frame sequence cannot be displayed at reasonable quality without those encoded parts.

## 8 File systems

```
several concurrent stream (video-on-demand) -> server chapter
concentrate on small number of streams (single-user workstation)

Rang9303:Techniques
Redd9308:Disk
Chen9308:Optimization
Gemm9308:Multimedia
Toba9308:Streaming
```

```
Baco9110:Multi
Yu8507:Efficient (Placement)
Oust8901:Beating
Lee9202:RAID
Tind93:Scheduling
Hart9204:Zebra
Delg9303:Media
Cabr9106:Exploiting
Loug9302:Continuous
Ande9108:Real
Rang9106:Testbed
Jord9208:Network
Rama9311:Operating (OS support for VOD)
```

# 9 Multimedia Operating Systems

## 9.1 Extensions of Commercial Operating Systems

Solaris scheduling; SGI?; Windows built-in DLLs

## 9.2 Mach Extensions

[46] [78] [40]

## 9.3 Real-Time and Multimedia Operating Systems

[79] (ARTS) [80] (Rialto) [81] (RT) [82] [83] (threads) [17]
set-top box systems (Kalevaras system?)

# 10 Conclusion

In this chapter, we have attempted an overview of the challenges of providing integrated continuous media on general-purpose workstations, focusing on issues of CPU scheduling, memory access and general operating system enhancements. Related subjects include the implementation of network protocols (Chapter XXX) and storage architectures for media playback (Chapter XXX). Also, video-on-demand servers are likely to have somewhat different requirements in that their workload is more predictable, primarily because they are mainly dispensing bits rather than processing them.

# References

[1] A. S. Tanenbaum, *Operating systems: design and implementation*. Englewood Cliffs, New Jersey: Prentice-Hall, 1987.

[2] J. L. Peterson and A. Silberschatz, *Operating System Principles*. Reading, Massachusetts: Addison-Wesley, 1983.

[3] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*. Reading, Massachusetts: Addison-Wesley, 1988.

[4] M. J. Bach, *The Design of the Unix Operating System*. Englewood Cliffs, New Jersey: Prentice-Hall, 1986.

[5] D. P. Anderson, S.-Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for continuous media in the DASH system," Technical Report CSD 89/537, University of California, Berkeley, Oct. 1989.

[6] R. Schaufler, "Realtime workstation performance for MIDI," in *Proceedings of the Winter USENIX Conference*, (San Francisco, California), pp. 139–151, Usenix, Jan. 1992.

[7] P. G. Milazzo, "Shared video under Unix," in *Proceedings of the Summer USENIX Conference*, (Nashville, Tennessee), pp. 369–383, June 1991.

[8] D. B. Terry and D. C. Swinehart, "Managing stored voice in the Etherphone system," *ACM Transactions on Computer Systems*, vol. 6, pp. 3–27, Feb. 1988.

[9] B. Arons, C. Binding, K. Lantz, and C. Schmandt, "The VOX audio server," in *Multimedia '89: 2nd IEEE COMSOC International Multimedia Communications Workshop*, (Ottawa, Ontario), Apr. 1989.

[10] G. D. Ripley, "DVI – a digital multimedia technology," *Communications ACM*, vol. 32, pp. 811–822, July 1989.

[11] L. F. Ludwig and D. F. Dunn, "Laboratory for the emulation and study of integrated and coordinated media communication," in *Proc. of ACM Sigcomm*, (Stowe, Vermont), pp. 283–291, Aug. 1987.

[12] A. Hopper, "Pandora – an experimental system for multimedia applications," *ACM Operating Systems Review*, vol. 24, pp. 19–34, Apr. 1990. also as Olivetti Technical Report.

[13] Parallax Graphics, "The Parallax 1280 series videographic processor," technical report, Parallax Graphics, 1987.

[14] I. M. Leslie, D. R. McAuley, and D. L. Tennenhouse, "ATM everywhere?," *IEEE Network*, vol. 7, pp. 40–46, Mar. 1993.

[15] D. R. McAuley, "Operating system support for the Desk Area Network," in *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, (Lancaster, U.K.), pp. 3–10, Lancaster University, Nov. 1993.

[16] M. Hayter and D. McAuley, "The desk area network," *ACM Operating Systems Review*, vol. 25, pp. 14–21, May 1991.

[17] I. M. Leslie, D. R. McAuley, and S. J. Mullender, "Pegasus — operating system support for distributed multimedia systems," *ACM Operating Systems Review*, vol. 27, pp. 69–78, Jan. 1993.

[18] C. J. Lindblad, D. J. Wetherall, W. F. Stasior, J. F. Adam, H. H. Houh, M. Ismert, D. R. Bacher, B. M. Philips, and D. L. Tennenhouse, "Distributed video applications – a software oriented approach," in *Gigabit Networking Workshop (GBN)*, (Toronto, Canada), IEEE, June 1994.

[19] K. Patel, B. C. Smith, and L. A. Rowe, "Performance of a software MPEG video decoder," in *Proceedings of ACM Multimedia 93*, (Anaheim, CA), Aug. 1993.

[20] Sun Microsystems, *SunVideo User's Guide*. Sun Microsystems, Mountain View, California, Aug. 1994.

[21] R. Frederick, "Experiences with real-time software video compression," in *Sixth International Workshop on Packet Video*, July 1994.

[22] T. Turletti, "The INRIA videoconferencing system IVS," *Connexions*, vol. 8, pp. 20–24, Oct. 1994.

[23] H.-C. Huang, J.-H. Huang, and J.-L. Wu, "Real-time software-based video coder for multimedia communication systems," in *Proceedings of ACM Multimedia '93*, (Anaheim, California), pp. 65–73, Aug. 1993.

[24] H. Schulzrinne, "Voice communication across the Internet: A network voice terminal," Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.

[25] W. Tawbi, F. Horn, E. Horlait, and J.-B. Stéfani, "Video compression standards and quality of service," *Computer Journal*, vol. 36, pp. 43–54, Feb. 1994.

[26] J. K. Ousterhout, "Why aren't operating systems getting faster as fast as hardware?," in *Proceedings 1990 Summer USENIX Conference*, (Anaheim, California), June 1990. Code archived at ftp://sprite.cs.berkeley.edu/bench.tar.Z.

[27] D. P. Anderson and R. Wahbe, "A framework for multimedia communication in a general-purpose distributed system," Tech. Rep. UCB/CSD 89/498, University of California at Berkeley, CS Division, Berkeley, California, Mar. 1989.

[28] A. Pearl, "System support for integrated desktop video conferencing," Technical Report TR-92-4, Sun Microsystems Laboratories, Mountain View, California, Dec. 1992.

[29] Ç. a. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communications in packet-switched networks," *Proceedings of the IEEE*, vol. 82, pp. 122–139, Jan. 1994.

[30] M. B. Jones, "Adaptive real-time resource management supporting modular composition of digital multimedia services," in *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, (Lancaster, U.K.), pp. 11–18, Lancaster University, Nov. 1993.

[31] R. G. Herrtwich, "The role of performance, scheduling, and resource reservation in multimedia systems," in *Operating Systems of the 90s and Beyond* (A. Karshmer and J. Nehmer, eds.), (Dagstuhl Castle, Wadern, Germany), pp. 279–284, July 1994. Springer Lecture Notes in Computer Science 563.

[32] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, pp. 114–131, Jan. 1991.

[33] R. L. Cruz, "A calculus for network delay, part II: Network analysis," *IEEE Transactions on Information Theory*, vol. 37, pp. 132–141, Jan. 1991.

[34] D. P. Anderson, "Meta-scheduling for distributed continuous media," Technical Report UCB/CSD 90/599, University of California, Berkeley, Oct. 1990.

[35] R. Govindan, "Operating systems mechanisms for continuous media," Tech. Rep. UCB/CSD 92/697, University of California, Berkeley, July 1992.

[36] R. Terek and J. Pasquale, "Experiences with audio conferencing using the X window system, UNIX, and TCP/IP," in *Proceedings of the Summer USENIX Conference*, (Nashville, Tennessee), pp. 405–418, June 1991.

[37] J. Nieh, J. G. Hanko, J. D. Northcutt, and G. A. Wall, "SVR4 UNIX scheduler unacceptable for multimedia applications," in *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, (Lancaster, U.K.), pp. 35–47, Lancaster University, Nov. 1993.

[38] S. McCanne and C. Torek, "A randomized sampling clock for CPU utilization estimation and code profiling," in *Proc. of Winter USENIX Conference*, (San Diego, California), Jan. 1993.

[39] T. Sanderson, S. Ho, N. Heijden, E. Jabs, and J. L. Green, "Near-realtime data transmission during the ICE comet Giacobini-Zinner encounter," *ESA Bulletin*, vol. 45, no. 21, 1986.

[40] H. Tokuda, T. Nakajima, and P. Rao, "Real-time Mach: Towards a predictable real-time system," in *Proc. of USENIX Mach Workshop*, (Burlington, Vermont), Oct. 1990.

[41] S. Khanna, M. Sebrée, and J. Zolnowsky, "Realtime scheduling in SunOS 5.0," in *Proceedings of Winter 1992 USENIX Conference*, 1992.

[42] O. Hagsand and P. Sjödin, "Workstation support for real-time multimedia communication," in *Proceedings of the Winter USENIX Conference*, (San Francisco, California), pp. 133–142, Usenix, Jan. 1994.

[43] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-time synchronization protocols for multiprocessors," in *Proceedings of the 1988 Real-Time Systems Symposium*, 1988.

[44] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, pp. –, Sept. 1990.

[45] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, Jan. 1973.

[46] J. Nakajima, M. Yazaki, and H. Matsumoto, “Multimedia/realtime extensions for the Mach operating system,” in *Proceedings of the Summer USENIX Conference*, (Nashville, Tennessee), pp. 183–198, June 1991.

[47] K. Jeffay, “On kernel support for real-time multimedia applications,” in *Proceedings Third IEEE Workshop on Workstation Operating Systems*, (Key Biscayne, Florida), pp. 39–46, Apr. 1992.

[48] K. Jeffay, D. Stone, and D. Poirier, “Yartos: kernel support for efficient, predictable real-time systems,” in *Real-Time Programming* (W. A. Halang and K. Ramamritham, eds.), pp. 7–12, Pergamon Press, 1992.

[49] S. Evans, K. Clarke, D. Singleton, and B. Smaalders, “Optimizing Unix resource scheduling for Unix interaction,” in *Proceedings of USENIX Summer 1993 Technical Conference*, (Cincinnati, Ohio), pp. 205–218, USENIX, June 1993.

[50] K.-J. Lin, S. Natarajan, and J. W.-S. Liu, “Imprecise results: utilizing partial computations in real-time systems,” in *Proc. of the Eighth IEEE Real-Time Systems Symposium*, (San Jose, California), pp. 210–217, Dec. 1987.

[51] J. K. Dey Sircar, J. F. Kurose, and D. Towsley, “On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks,” *Submitted to IEEE Transactions on Computers*, vol. ??, pp. –, June 1994.

[52] J. K. Dey, J. F. Kurose, D. Towsley, and M. Girkar, “Efficient on-line processor scheduling for a class of IRIS (increasing reward with increasing service) real-time tasks,” in *Proc. of ACM Sigmetrics Conference*, pp. 217–228, May 1993.

[53] J. G. Hanko, E. M. Kuerner, J. D. Northcutt, and G. A. Wall, “Workstation support for time-critical applications,” in *Proc. of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. –, Nov. 1991.

[54] S. Ramanathan and P. V. Rangan, “Feedback techniques for intra-media continuity and inter-media synchronization in distributed multimedia systems,” *Computer Journal*, vol. 36, pp. 19–31, Feb. 1994.

[55] S. Ramanathan and V. P. Rangan, “Adaptive feedback techniques for synchronized multimedia retrieval over integrated networks,” *IEEE/ACM Transactions on Networking*, vol. 1, pp. 246–260, Apr. 1993.

[56] D. C. A. Bulterman, “Synchronization of multi-sourced multimedia data for heterogeneous target systems,” in *Third International Workshop on network and operating system support for digital audio and video*, (San Diego, California), pp. 110–120, IEEE Communications Society, Nov. 1992.

[57] T. D. C. Little, A. Ghafoor, C. Y. R. Chen, C. S. Chang, and P. B. Berra, “Multimedia synchronization,” *The Quarterly Bulletin of the IEEE Computer Society Technical Committe on Data Engineering*, vol. 14, pp. 26–35, Sept. 1991.

[58] D. L. Mills, “Internet time synchronization: the network time protocol,” *IEEE Transactions on Communications*, vol. 39, pp. 1482–1493, Oct. 1991.

[59] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, “Adaptive playout mechanisms for packetized audio applications in wide-area networks,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Toronto, Canada), June 1994.

[60] AT&T, *Streams Programmers Guide and Streams Primer*. Englewood Cliffs, New Jersey: Prentice Hall, 1987.

[61] S. McCane and V. Jacobson, “A BSD packet filter: A new architecture for user-level packet capture,” in *Proceedings of the Winter USENIX Conference*, (San Diego, California), pp. 259–269, Usenix, Jan. 1993.

[62] M. Pasieka, P. Crumley, A. Marks, and A. Infortuna, “Distributed multimedia: how can the necessary data rates be supported?,” in *Proceedings of the Summer USENIX Conference*, (Nashville, Tennessee), pp. 169–182, June 1991.

[63] F. O. Hoffmann and L. Delgrossi, “ST-II for the Heidelberg transport system: protocol, design and implementation.” technical memorandum, Sept. 1993.

[64] A. Wolman, G. Voelker, and C. A. Thekkath, “Latency analysis of TCP on ATM network,” in *Proceedings of the Winter USENIX Conference*, (San Francisco, California), pp. 167–179, Usenix, Jan. 1994.

[65] W. W. Gaver, “Sound support for collaboration,” in *Proceedings of the Second European Conference on Computer-Supported Cooperative Work (ECSCW’91)* (L. Bannon, M. Robinson, and K. Schmidt, eds.), (Amsterdam, The Netherlands), pp. 293–308, Sept. 1991.

[66] H. A. Chinn, D. K. Gannett, and R. M. Morris, “A new standard volume indicator and reference level,” *Bell System Technical Journal*, vol. 19, pp. 94–137, Jan. 1940.

[67] S. Angebranndt, R. L. Hyde, D. H. Luong, N. Siravara, and C. Schmandt, “Integrating audio and telephony in a distributed workstation environment,” in *Proceedings of the Summer USENIX Conference*, (Nashville, Tennessee), pp. 419–435, June 1991.

[68] G. V. Neville-Neil, “Current efforts in client/server audio,” *X Resource*, vol. 8, pp. 69–86, 1992.

[69] J. Fulton and G. Renda, “The network audio system,” in *X Technical Conference*, X Consortium, 1994. also in *X Resource*, Issue 9, pp. 181.

[70] T. M. Levergood, A. C. Payne, G. James, G. W. Treese, and L. C. Stewart, “AudioFile: A network-transparent system for distributed audio applications,” Technical Report 93/8, Digital Equipment Corporation, Cambridge Research Lab, Cambridge, Massachusetts, June 1993.

[71] T. M. Levergood, A. C. Payne, J. Gettys, W. G. Treese, and L. C. Stewart, “Audiofile: a network-transparent system for distributed audio applications,” in *Proceedings of USENIX Summer Technical Conference*, (Cincinnati, Ohio), pp. 219–236, USENIX, June 1993.

[72] P. Schnorf, "Integrating video into an application framework," in *Proc. of ACM Multimedia*, (Anaheim, California), pp. 411–417, Aug. 1993.

[73] Sun Microsystems, *XIL Programmer's Guide*. Sun Microsystems, Mountain View, California, Aug. 1994.

[74] E. Rennison, R. Baker, D. D. Kim, and Y.-H. Lim, "MuX: an X co-existent time-based multimedia I/O server," *The X Resource*, vol. 1, pp. 213–233, Winter 1992.

[75] C. Zimmermann, "Making distributed multimedia systems secure: the switchboard approach," *ACM Operating Systems Review*, vol. 28, pp. 88–100, Jan. 1994.

[76] C. C. Center, "Cert advisory ca-93:15: /usr/lib/sendmail, /bin/tar, and /dev/audio vulnerabilities," Oct. 1993.

[77] J. C. Tang and E. A. Isaacs, "Why do users like video? studies of multimedia-supported collaboration," Technical Report TR-92-5, Sun Microsystems Laboratories, Mountain View, California, Dec. 1992.

[78] K. Kawachiya, M. Ogata, N. Nishio, and H. Tokuda, "Evaluation of QOS-Control servers on real-time mach," in *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, (Durham, New Hampshire), pp. 123–126, Apr. 1995.

[79] H. Tokuda and C. W. Mercer, "ARTS: A distributed real-time kernel," *ACM Operating Systems Review*, vol. 23, pp. 29–53, July 1989.

[80] M. Jones, P. Leach, R. Draves, and J. Barrera, "Support for user-centric modular real-time resource management in the rialto operating system," in *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, (Durham, New Hampshire), pp. 55–66, Apr. 1995.

[81] J. A. Stankovic and K. Ramamritham, "The spring kernel: A new paradigm for real-time operating systems," *Operating Systems Review*, vol. 27, pp. 54–71, July 1989.

[82] S. J. Mullender, "Systems for the nineties – distributed multimedia systems," in *Operating Systems of the 90s and Beyond* (A. Karshmer and J. Nehmer, eds.), (Dagstuhl Castle, Wadern, Germany), pp. 273–278, July 1994. Springer Lecture Notes in Computer Science 563.

[83] K. Schwan, H. Zhou, and A. Gheith, "Real-time threads," *ACM Operating Systems Review*, vol. 25, pp. 35–46, Oct. 1991.