Multimedia Systems (1996) 4: 269–280

Multimedia Systems

Operating system issues for continuous media

Henning Schulzrinne*

GMD-Fokus, Hardenbergplatz 2, 10623 Berlin, Germany

Abstract. Continuous media such as audio and video pose new challenges to all parts of multipurpose operating systems. We discuss issues related to CPU scheduling, memory allocation, system support and application environments and summarize some of the solutions proposed in the literature.

Key words: Continuous media – Operating system enhancements – CPU scheduling – Network implementation

1 Introduction

Operating systems play a crucial role in supporting multimedia applications. Here we 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 a 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 paper, with a strong emphasis on implementation-related work rather than on 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 must 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. For simplicity, we include "recording", as for a tape recorder, when using the term "playback". Video conferencing is the stereotypical application for interactive continuous media. For interactive continuous media, the acceptable delay is often governed by human reaction time or issues of echo for audio. These delays are in 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 various media within the same system, some of which may well be continuous media of both playback and interactive types, but also can be still images, text, or graphics. One specific problem is the coordination in time (synchronization) of these media.

Because their characteristics differ most markedly from traditional data types supported within operating systems, we focus largely on continuous media here. It is often assumed that continuous media are challenging because they impose the highest data rates or processing on the system. This is not necessarily 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 are 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 do not touch the operating system or main memory, but rather use 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 those in [7] had the workstation control a VCR or laser disk player, with video either displayed on a separate monitor or fed through

* e-mail: schulzrinne@fokus.gmd.de

an analog mixer. For a monochrome system, a simple control plane can be used, while for a color system, chromakeying is a traditional technique long used – as 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 volume unit (VU) display cannot readily be 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] integrates video in digital form. This is done through a separate video processor and a pixel switch that decides 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 treating continuous media as data to be either processed or simply switched from, say, network device to display device. If data do not touch the CPU on their path through the system, operating system requirements are much relaxed. Thus, for our purposes, we do not discuss this mode further.

Given that many continuous media applications need not pass media data through the CPU (with one of the approaches outlined), the reader may ask why integration is desirable at all, given the complexity.

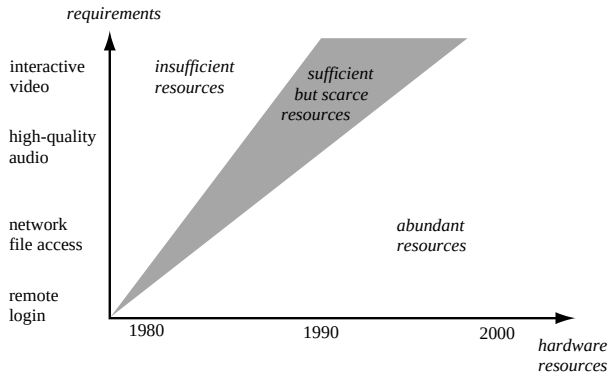
Naturally, AI-like functions like image categorization require CPU intervention, but other, lower-level operations like encoding and decoding, scaling, or sampling rate conversion can be done either in hardware or software. Software generally has a price advantage. In particular, the incremental cost 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 bypass the operating system and the 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], audio [24], and 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. 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.

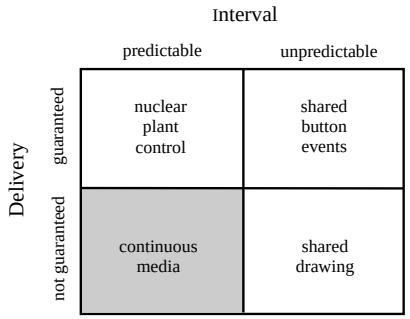
While the increasing computational power of workstations should enable fairly extensive manipulation of continuous media, a number of factors reduce the ability of workstations to deal with continuous media below what might be expected from pure MIPS or SPECmark benchmark numbers. Continuous media stress a number of architectural areas poorly covered by traditional CPU-oriented benchmarks and areas in which 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 μ-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/s.) MIDI has a low data rate of 31 kbits/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 Mbits/s. Video exacerbates the demands by requiring data rates of about 1.2 Mbits/s for MPEG-compressed, VHS-quality video [25], 20 Mbits/s for compressed HDTV, up to 200 Mbits/s for 24-bit (RGB) uncompressed video at 30 frames/s (640 by 480 pixels) or 1.2 Gbits/s for uncompressed HDTV. While networks and file servers 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 have not improved dramatically either. Disk caches are of limited use since most



1



2

Fig. 1. The window of scarcity [27]

Fig. 2. Characteristics of time-critical data [28]

continuous media are 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 as in 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 labeled abundance of resources on the right, we may have enough resources on average for all comers, but may still have to maintain local control to avoid missing deadlines for individual streams. Note that this division corresponds with those in communication resources. There, the window of scarcity reflects the region in which admission control is needed.

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

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. It must 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 and protocols for data sharing and by orchestrating multimedia data.

In the following sections, we discuss two of the major bottleneck system resources, CPU and disk. Then we look at higher-level system support through libraries and at security issues. Finally we discuss some systems explicitly designed for continuous media applications. We 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 those of a distributed network. Furthermore 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 as in 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, must be processed reliably within a few hundred milliseconds, but do not have predictable arrival patterns.

2.2 Workload characterization and admission control

Current timesharing operating systems 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 quality-of-service QOS guarantees unless the traffic is bounded in burstiness and the 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 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 is required as CPU and memory management unit (MMU) caches are 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 based on prior execution have been made, e.g., by Jones [30]. However, as pointed out in [31], upper or pessimistic bounds may lead to poor resource utilization and unnecessary denials of service. However, incomplete resource utilization by continuous media processes 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 computer-supported cooperative work (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, and conference control. 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.

Many interactive, continuous media applications are characterized by a cycle, occurring at frame rate or audio packet rate, and 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, 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 a strong dependency on image content and frame size. 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 for 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 pre-empted, 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, 34–36] 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.

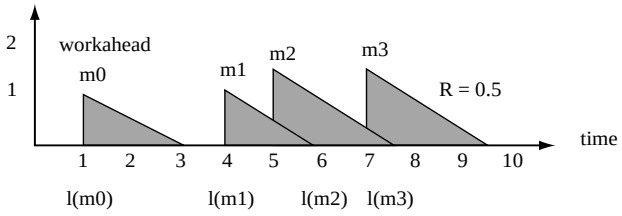


Fig. 3. Example of workahead function [5]

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 message arriving late can delay the logical arrival time of all subsequent messages and thus their processing. Also, the burst size B does not enter into the computation of the logical arrival time. A message arriving after its logical arrival time is termed *critical*, while one arriving before is termed *workahead*. An example is shown in Fig. 3.

Any of the scheduling algorithms discussed in this section apply to both processes and threads. (Threads are schedulable entities without their own address space.) Indeed, Govindan and Anderson argue, in their split-level scheduling proposal [37], that both real-time threads and processes should be supported. Scheduling within threads, if implemented at the user level rather than in the kernel, has the advantage of avoiding some of the frequent context switches and user-kernel interactions.

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 cannot provide 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. (This accounts for the success of multimedia demos at trade shows.) 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 [38].

The reasons for these variations are found in the properties of kernels and process schedulers. Both System-V [4] and Berkeley software distribution (BSD)-derived schedulers [3, p. 86] use a multilevel feedback queue, where processes

are served in round robin fashion 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 pre-empt it when starting or if they can "convince" the kernel that they are more I/O intensive. (Process priority is highest at the time the process is created, unless manually changed by the superuser.) A process is only pre-empted after using its time slice 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 was 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 [39]. They discuss how a video application, a typing simulator, the X server, and a compiler-like computing 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 priority of a process 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 is charged the full CPU tick or nothing at all [40], 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 [41, 42] generally implement fixed priority, pre-emptive schedulers and are thus not suited for a general-purpose system shared by real-time (continuous media) interactive and batch applications.

2.4 Static priority for real-time processes and threads

There are a number of conceptually simple ways of improving the behaviour of standard operating systems like UNIX 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 [43] and the Mach operating system implement a thread-based, fully-pre-emptible 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 pre-emption points [44], where system routines such as fork, exec, or exitcheck 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 must execute semaphores and the run queue must be checked at every pre-emption point.

UNIX SVR4 also introduced installable scheduling classes [6, 43], including a real-time class with fixed priorities (which are always higher than those of processes within the times-haring class). Processes in the real-time class can execute until either blocked or finished, or a time limit can be associated with the process. Since the priority of the real-time class must 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. (Not surprisingly, use of this class requires superuser privileges.) This is documented in [39] through the example of a video display process that can cause the whole system to become unresponsive. However, the example chosen in [39] 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 [39] present a new timesharing class that provides a better balance among 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 must rely on other system services, that are likely to execute as times-haring or system processes. Examples include the X-window server for video output and the stream handler for audio input and output. To avoid wholesale priority elevation of all system services, they must 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 thread libraries, however, are only scheduled within the time slice of the overall process.

To allocate CPU cycles to interactive continuous media processes, results from the scheduling of hard real-time processes can be used. (A hard real-time system suffers catastrophic failure if a process is not completed before the assigned deadline. Examples include air-traffic control, process control, and embedded systems [45].) A simple, static priority policy that is widely used in such systems is the *rate-monotonic* scheduler [46]. The policy is pre-emptive and assigns scheduling priorities according to the rate of arrival for a particular kind of process, with higher priorities for more frequent processes. In order for processes to meet their deadlines, the schedulability must be tested prior to allowing a new process to use CPU resources. For processes that are periodic, independent, and have a deadline equal to their period, there are necessary and sufficient conditions that ensure this.

Liu and Layland show that, for periodic, noncommunicating processes with constant computation times and zero context switch overheads a set of processes can successfully be scheduled if

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

where n is the number of periodic programs, with program i requiring C_i of computation at a period of T_i. For a large n, this bounds the utilization from below to about 69%. However, Lehoczky et. al [47] show that, on average, a utilization of 88% can be reached. Others [48, 49] have provided bounds requiring fewer assumptions, though they may be harder to compute.

While the Liu/Layland assumptions are onerous for many hard real-time systems, they reflect the processing of continuous media reasonably well. If anything, the delay bound of a single period is probably excessive, as audio and video can typically tolerate delay jitter higher than a single period if the output device provides sufficient buffering. A generalization of the rate-monotonic policy to is the *deadline-monotonic* [50] scheduler, with priorities assigned according to process deadlines. These deadlines do not have to be equal to the period. Both are optimal static priority policies in that if any static priority policy can schedule a set of processes, the rate and deadline-monotonic ones can as well. A schedulability test is presented in [45], while a general survey of scheduling algorithms is presented by Audsley and Burns [51].

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 pre-empted by a medium-priority process, delaying release of the resource and blocking the high-priority process for an indeterminate time [52]. There are a number of remedies [48], 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 [43].

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 closer deadlines. However, this pre-emptive 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 assign priorities manually, we can schedule processes with the earliest deadlines first (EDF) [46]. EDF scheduling does not require processes to be periodic; it also has nice optimality properties in that it yields a schedule that avoids missed deadlines if at all possible. EDF scheduling is also used in communication networks, but for CPU scheduling, the pre-emptive variety is appropriate, with the readiness of a new process with a closer deadline pre-empting the currently executing process. Liu and Layland show that, for periodic tasks,

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

is sufficient for schedulability.

EDF scheduling within the UNIX operating system has been implemented by Hagsand and Sjödin [44], for Mach by Tokuda et. al [42] and Nakajima et. al [53]. A more sophisticated version of EDF scheduling, called processor capacity reserves [54, 55], was added to real-time Mach [42]. Here, programs are assigned a periodic processor capacity, with programs that have not yet consumed their allocation taking precedence over unreserved programs. In a microkernel environment, particular care must be taken to ensure that all user-level server threads invoked by the kernel thread are "charged" to the invoking thread.

In [44], a process starts towards its deadline if one of the designed file descriptors becomes 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 from each other.

The Yet Another Real-Time Operating System (YARTOS) [56, 57] uses a related approach. Here, the deadline clock starts to tick with the arrival of messages (events), a generalization of the file-descriptor approach already mentioned. 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, 36]. With the LBAP workload characterization described in Sect. 2.2, a simple policy can be formulated: critical processes are scheduled with the EDF; regular interactive (non-real-time) processes have priority over workahead processes, but are pre-empted when such 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 approach , 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 [58]. Making class assignments accessible to the superuser 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 in which 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 [59]. 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 tasks are labeled as *imprecise computations*

or are said to give increasing reward with increasing service (IRIS). In principle, imprecise computation offers a graceful degradation of the media quality within a continuous media system as the system load increases.

This seemingly straightforward and appealing approach has a practical problem. The typical continuous media computation consists of a CPU-intensive task that can be varied in length and is followed by "packaging" (e.g., Huffman coding for JPEG, or generating MPEG frames) and any system overhead needed to render or transfer the processed data. While there are theoretical investigations in the properties of inprecise computation [60, 61], 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 time scale. Instead of rejecting new real-time tasks, the system detects overload and then renegotiates resource requirements with these tasks [62]. A real-time thread package for Mach [63] supports both periodic and aperiodic execution with deadline bounds. It uses timing-fault notification for threads serving adaptive applications to self-stabilize. A timing-fault occurs if a thread misses its deadline. The timing fault handler can then reduce the requested frequency of thread execution to self-stabilize the system. With the right control loop, this ensures smooth display of video sequences, if at a reduced rate, under the conditions of a heavy load. An alternate approach explored by the authors has a QOS manager control the frame rate within bounds specified by the thread.

Unlike most computational (batch) applications that are useful over a range of execution durations from milliseconds to hours, most continuous media applications have a fairly narrow range of adaptability. Video frame rates can perhaps be varied by a factor of 2 (from 30 to 15 frames/s say) without fundamentally being 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-bit-rate coding would likely be more computationally intensive. Thus, network-motivated adjustments must be carefully coordinated with local system resources.

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 in 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 the initial load during start-up.

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 [58]. Instead of the relatively small number of active processes when user interaction 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 time-sharing 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 singly-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 a process telling the linker to colocate certain memory segments could lead to decreased working sets [58], but are 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 less than 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 graeter 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 of pre-emption 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 [64, 65, 66, 67]. It appears, however, that once synchronized clocks are available with clock differences of a few milliseconds [68], the problem is largely solved. Explicit synchronization algorithms appear to provide special case, multiparty 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 [69].

4.2 Interrupt handling

As mentioned in Sect. 1, some continuous media types may cause high system load, not due to their bandwidth, but rather to 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 pre-emptive multitasking and separate address spaces (e.g., Microsoft Windows 3.1 or Apple System 7) may perform significantly better in real-time 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, nonpredictable delays for other processes [6]. One approach for System-V-derived kernels may be the use of modules pushed onto the streams-processing stack [70], 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 [71] 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 that 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”, pre-empt 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 that of all other interrupt handlers, but still pre-empting 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 them [44]. However, deciding whether, for example, a packet belongs to a network association deserving high-priority treatment may not be trivial at hard interrupt time and could outweigh any gains. As is the case 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 or 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 [72]. 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, in which only a special memory region can be remapped and only between the same virtual addresses in sender and receiver address spaces. A related approach presented in [37] emphasizes shared memory for synchronizing data transfer between the user and the kernel space rather than (necessarily) for the actual data transfer.

5 Network interface and protocol processing

High-rate continuous media pose some of the same performance problems as volume data applications. Solutions include the elimination of copying between network adapter [73] and operating system buffers, or kernel and user space, or combining copying and checksumming [74]. For continuous media, the User Datagram Protocol (UDP) is used as a transport protocol rather than the Transmission Control Protocol (TCP), as the reliability and flow-control offered by TCP interfere with the delay requirements and inherent rate of continuous media. However, UDP checksumming still imposes unnecessary processing costs. Measurements [74] have shown that, for large packets as would be found in video transmission, the sender latency is dominated by checksumming and, if used, any software fragmentation into ATM cells. For example, for 1400-byte packets, a BSD 4.4 sender implementation spends 39% of its time on the TCP checksum (which is the same as the UDP checksum). For audio and video, bit errors are usually preferable to dropping packets. For video, even if the bit error corrupts the frame encoding structure, at least the part of the video frame preceeding the bit error can be used by the receiver. Unfortunately, most existing network stacks do not allow checksumming to be turned off for an individual network association. (Turning off UDP checksums in general can lead to corrupted data in the network file system.)

For the small packets typically found for low-bit-rate voice, processing costs are more evenly distributed among copies, checksum, protocol processing, operating system-priority management, the network-interface driver and interrupt handling [75]. For low-latency voice, packets arrive at a frequency of at least 50 packets/s. Thus, efficient interrupt handling and process scheduling is particularly important (see also Sect. 4.2). Even with a relatively slow workstation, however, per-packet processing delay was measured at less than 500 μs per packet for TCP and should be significantly less for UDP.

Protocols like ST-II [76] claim to be particularly optimized for multimedia streams, offering both facilities for resource reservation, and, less importantly, a reduced per-packet handling overhead. However, it appears [77] that the performance gains are modest.

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 his own audio and video programming interface. Generally, both audio and video are sequential devices, so that the basic UNIX device or streams model is appropriate, enhanced with the necessary control functionality for adjusting parameters such as the sampling rate, 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, nonshareable 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 first-in, first out (FIFO) buffer before the kernel signals to the application that a new chunk of audio data is ready for reading. This would mean that interactive applications could trade low delay for occasional audio glitches, while playback and recording applications could incur a higher delay and fewer context switches. Furthermore, 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 [78] are difficult to implement this way.

An audio API (or engine) should allow several applications to share the single speaker by either mixing at various volumes or priority pre-emption. External applications like VU meters [79], 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 the 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 LAN. Thus, a general system solution seems preferable to 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, the audio library must provide either almost all the desirable audio services or very few beyond mixing, volume control and the like.

Existing implementations, such as those discussed in [38] 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 by mixing them in some user-determined weighting.

The Alib client-server library [80] allows to be tied together by logical “wires”. Such components include telephones, audio input and output devices, speech synthesizers and recognizers, recorders, and playback devices through logical “wires”. Commands, sent through a separate command pipeline, are automatically sequentialized, so that a playback operation can be followed immediately by recording audio.

Some attempts have been made at cross-platform APIs with enhanced functionality[81], but these appear to have a number of shortcomings. Most audio APIs seem to be designed mainly for playing back and recording audio clips rather than for real-time use. Netaudio [82] and AudioFile [83, 84] 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 separate control and data streams. 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 devices controlled 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.
- It should be possible either to place the client and server within the same LAN, with low latency and delay jitter, or to spread then across a wide area network (WAN), with large delay jitter and packet loss. In other words, the transport mechanism cannot be a traditional data transport protocol; adaptive compensation of playout delay needs to be handled.
- Streams should be able to share physical audio devices, by pre-emption, mixing, or a combination (lowering volume of other streams).

– Servers should cache sound clips for repeated play.
– Resource reservation may be needed to ensure that decoding and mixing does not exceed the 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 [85] describes an extension of the ET++ windowing toolkit, in which motion video, generated by a hardware or software decoder, becomes a first class object that can to be clipped and moved like other windowing objects. The XIL [86] 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 on what operations can be combined, it is easy to generate sequences in which 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 [87] 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 follow 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 a workstation, the workstation can become a remotely controlled monitoring device [28]. For example, some workstations were delivered with the audio input device set world-readable, allowing remote listening to whomever attached a process to that device [88, 89]. 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 [90] 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, the use of receiver-oriented multicasting for efficiency will make encryption the only viable means of protecting privacy. Zimmermann [88] suggests a capability-based protection system for multimedia operating systems.

If continuous media data are 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. 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 Conclusion

We have attempted to offer 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, with a brief overview of network implementation issues. The survey has ignored issues of storage scheduling [91] and management [92], as they are generally of less importance for end-user workstations. These issues dominate, however, for video-on-demand servers [93]. It is also not yet clear whether general-purpose workstations and operating systems are the most suitable for large-scale servers, particularly if they are to generate constant rate, synchronous streams for set-top boxes. However, many of the approaches used by the network file system (NFS) and web servers might also be applicable here. 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.

General purpose, multiuser operating systems can still only deliver relatively small amounts of video information, with very limited processing, even though raw CPU and bus performance would seem to allow much better capabilities. The theoretical issues seem well understood, but implementations need to follow suit.

References

1. Tanenbaum AS (1987) Operating systems: design and implementation. Prentice-Hall, Englewood Cliffs
2. Peterson JL, Silberschatz A (1983) Operating system principles. Addison-Wesley, Reading, Mass
3. Leffler SJ , McKusick MK, Karels MJ, Quarterman JS, (1988) The design and implementation of the 4.3BSD UNIX operating system. Addison-Wesley, Reading, Mass
4. Bach MJ (1986) The design of the UNIX operating system. Prentice-Hall, Englewood Cliffs
5. Anderson DP, Tzou S-Y, Wahbe R, Govindan R, Andrews M (1989) Support for continuous media in the DASH system. Technical Report CSD 89/537, University of California, Berkeley, Calif
6. Schaufler R (1992) Realtime workstation performance for MIDI. Proceedings of the Usenix Winter Conference , San Francisco, Calif, Usenix, Berkeley, CA, USA, pp 139–151

7. Milazzo PG (1991) Shared video under UNIX. in Proceedings of the Usenix Summer Conference, Nashville, Tenn., Usenix, Piscataway, NJ, USA, pp 369–383
8. Terry DB, Swinehart DC (1988) Managing stored voice in the Etherphone system. ACM Trans Comput Syst 6 :3–27
9. Arons B, Binding C, Lantz K, Schmandt C (1989) The VOX audio server. Multimedia '89: 2nd IEEE COMSOC International Multimedia Communications Workshop, Ottawa, Ontario, pp 1–23
10. Ripley GD (1989) DVI – a digital multimedia technology. Communications ACM 32:811–822
11. Ludwig LF, Dunn DF (1987) Laboratory for the emulation and study of integrated and coordinated media communication. SIGCOMM Symposium on Communications Architectures and Protocols, Stowe, Vt., ACM, pp 283–291
12. Hopper A (1990) Pandora – an experimental system for multimedia applications. ACM Operating Syst Rev 24:19–34 (also as Olivetti Technical Report.)
13. Parallax Graphics (1987) The Parallax 1280 series videographic processor. Technical Report, Parallax Graphics, Santa Clara, CA
14. Leslie IM, McAuley DR, Tennenhouse DL (1993) ATM everywhere?, IEEE Network 7:40–46
15. McAuley DR (1993) Operating system support for the desk area network. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, U.K., Lecture Notes in Computer Science 846, pp 13–20
16. M Hayter, McAuley D (1991) The desk area network. ACM Operating Syst Rev 25:14–21
17. Leslie IM, McAuley DR, Mullender SJ (1993) Pegasus – operating system support for distributed multimedia systems, ACM Operating Syst Rev 27:69–78
18. Lindblad CJ, Wetherall DJ, Stasior WF, Adam JF, Houh HH, Ismert M, Bacher DR, Philips BM, Tennenhouse DL (1994) Distributed video applications – a software oriented approach. Gigabit Networking Workshop (GBN), Toronto, Canada, IEEE
19. Patel K, Smith BC, Rowe LA (1993) Performance of a software MPEG video decoder. Proceedings of ACM Multimedia 93, Anaheim, Calif., ACM
20. Sun Microsystems (1994) SunVideo User's Guide. Sun Microsystems, Mountain View, Calif
21. Frederick R (1994) Experiences with real-time software video compression. The 6th International Workshop on Packet Video, ACM, September, Portland, OR, USA
22. Turletti T (1994) The INRIA videoconferencing system IVS. Connexions 8:20–24
23. Huang H-C, Huang J-H, Wu JL (1993) Real-time software-based video coder for multimedia communication systems. Proceedings of ACM Multimedia '93, Anaheim, Calif., ACM, pp 65–73
24. Schulzrinne H (1992) Voice communication across the Internet: a network voice terminal. Technical Report TR 92-50, Deptartment of Computer Science, University of Massachusetts, Amherst, Mass
25. Tawbi W, Horn F, Horlait E, Stéfani J-B (1994) Video compression standards and quality of service. Comput J 36:43–54
26. Ousterhout JK (1990) Why aren't operating systems getting faster as fast as hardware? Proceedings of the Usenix Summer Conference, Anaheim, Calif., Usenix, pp 247–256. Code archived at ftp://sprite.cs.berkeley.edu/bench.tar.Z
27. Anderson DP, Wahbe R (1989) A framework for multimedia communication in a general-purpose distributed system. Technical Report UCB/CSD 89/498, University of California at Berkeley, Computer Science Division, Berkeley, Calif
28. Pearl A (1992) System support for integrated desktop video conferencing. Technical Report TR-92-4, Sun Microsystems Laboratories, Mountain View, Calif
29. Aras CaM, Kurose JF, Reeves DS, Schulzrinne H (1994) Real-time communications in packet-switched networks. Proceedings of the IEEE 82:122–139
30. Jones MB (1993) Adaptive real-time resource management supporting modular composition of digital multimedia services. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, U.K., Lecture Notes in Computer Science 846, pp pp 21–28
31. Herrtwich RG (1994) The role of performance, scheduling, and resource reservation in multimedia systems. Operating Systems of the 90s and beyond, Wadern, Germany (Lecture Notes in Computer Science, Vol 563). Springer, Berlin Heidelberg New York, pp 279–284
32. Cruz RL (1991) A calculus for network delay. Part I: Network elements in isolation. IEEE Transactions on Information Theory 37:114–131
33. Cruz RL (1991) A calculus for network delay. Part II: Network analysis, IEEE Transactions on Information Theory 37:132–141
34. Anderson DP (1990) Meta-scheduling for distributed continuous media, Technical Report UCB/CSD 90/599, University of California, Berkeley, Calif
35. Govindan R (1992) Operating systems mechanisms for continuous media. Technical Report UCB/CSD 92/697, University of California, Berkeley, Calif
36. Anderson DP (1993) Metascheduling for continuous media. ACM Trans Comput Syst 11:226–252
37. Govindan R (1991) Anderson DP, Scheduling and IPC mechanisms for continuous media. Technical Report CSD-91-622, University of California at Berkeley, Berkeley, Calif
38. Terek R, Pasquale J (1991) Experiences with audio conferencing using the X window system, UNIX, and TCP/IP. Proceedings of the Usenix Summer Conference, Nashville, Tenn., Usenix, pp 405–418
39. Nieh J, Hanko JG, Northcutt JD, Wall GA (1993) SVR4 UNIX scheduler unacceptable for multimedia applications. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, U.K., Lecture Notes in Computer Science 846, pp 41–53
40. McCanne S, Torek C (1993) A randomized sampling clock for CPU utilization estimation and code profiling. Proceedings of the Usenix Winter Conference, San Diego, California, Usenix, pp 387–394
41. Sanderson T, Ho S, Heijden N, Jabs E, Green JL (1986) Near-realtime data transmission during the ICE Comet Giacobini-Zinner encounter. ESA Bulletin 45:21–23
42. Tokuda H, Nakajima T, Rao P (1990) Real-time Mach: Towards a predictable real-time system, in Proceedings of the Usenix Mach Workshop, (Burlington, Vermont), Oct., Usenix, pp 73–82
43. Khanna S, Sebrée M, Zolnowsky J (1992) Realtime scheduling in SunOS 5.0. Proceedings of the Usenix Winter Conference, Usenix, pp 375–390
44. Hagsand O, Sjdin P (1994) Workstation support for real-time multimedia communication. Proceedings of the Usenix Winter Conference, San Francisco, Calif., Usenix, pp 133–142
45. Audsley NC, Burns A, Richardson MF, Wellings AJ (1991) Hard real-time scheduling: The deadline monotonic approach. Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, Ga., IEEE 127–132
46. Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 20:46–61
47. Lehoczky J, Sha L, Ding Y (1989) The rate monotonic scheduling algorithm: Exact characterization and average case behavior. IEEE Real-Time Systems Symposium, Santa Monica, Calif., pp 166–171
48. Sha L, Rajkumar R, Lehoczky JP (1990) Priority inheritance protocols: an approach to real- time synchronization. IEEE Trans Comput 39:1175–1185
49. Sprunt B, Sha L, Lehoczky JP (1989) Aperiodic task scheduling for hard real-time systems, The Journal of Real-Time Systems, vol. 1, pp. 27–60, June
50. Leung JY-T, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation, 2:237–250
51. Audsley N, Burns A (1990) Real-time system scheduling. Technical Report YCS 134, University of York, first year report, ESPRIT BRA Project (3092).
52. Rajkumar R, Sha L, Lehoczky JP (1988) Real-time synchronization protocols for multiprocessors. Proceedings of the IEEE Real-Time Systems Symposium
53. Nakajima J, Yazaki M, Matsumoto H (1991) Multimedia/realtime extensions for the Mach operating system. Proceedings of the Usenix Summer Conference, Nashville, Tenn., Usenix, pp 183–198
54. Mercer CW, Savage S, Tokuda H (1993) Processor capacity reserves: an abstraction for managing processing usage. The 4th. Workshop on

Workstations Operating Systems, Napa, Calif., IEEE, pp 129–134
55. Mercer CW, Savage S, Tokuda H (1994) Processor capacity reserves: Operating system support for multimedia applications. Proceedings of the IEEE International Conference on Multimedia Computing and Systems. IEEE, pp 90–99 (This is a condensed version of Technical Report CMU-CS-93-157)
56. Jeffay K (1992) On kernel support for real-time multimedia applications. Proceedings of the 3rd IEEE Workshop on Workstation Operating Systems, Key Biscayne, Fla., IEEE, pp 39–46
57. Jeffay K, Stone D, Poirier D (1992) YARTOS: kernel support for efficient, predictable real-time systems. In: Halang WA, Ramamritham K (eds), Real-Time programming. Pergamon Press, New York, pp 7–12
58. Evans S, Clarke K, Singleton D, Smaalders B (1993) Optimizing UNIX resource scheduling for UNIX interaction. Proceedings of the Usenix Summer Conference, Cincinnati, Ohio, Usenix, pp 205–218
59. Lin K-J, Natarajan S, Liu JW-S (1987) Imprecise results: utilizing partial computations in real-time systems. Proceedings of the 8th IEEE Real-Time Systems Symposium, San Jose, Calif., IEEE, pp 210–217
60. Dey JK, Gicar M, Kurose JF, Towsley D (1994) On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. IEEE Trans Comput, to be published
61. Dey JK, Kurose JF, Towsley D, Girkar M (1993) Efficient on-line processor scheduling for a class of IRIS (increasing reward with increasing service) real-time tasks. Proceedings of the ACM Sigmetrics Conference, ACM, pp 217–228
62. Hanko JG, Kuerner EM, Northcutt JD, Wall GA (1991) Workstation support for timecritical applications. Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Lecture Notes in Computer Science 614, pp 4–9
63. Tokuda H, Kitayama T (1993) Dynamic QOS control based on real-time threads. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, U.K., Lecture Notes in Computer Science 846, pp 113–122
64. Ramanathan S, Rangan PV (1993) Feedback techniques for intra-media continuity and inter-media synchronization in distributed multimedia systems. Comput J 36:19–31
65. Ramanathan S, Rangan VP (1993) Adaptive Feedback techniques for synchronized multimedia retrieval over integrated networks, IEEE/ACM Trans Networking 1:246–260
66. Bulterman DCA (1992) Synchronization of multi-sourced multimedia data for heterogeneous target systems. The 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, Calif., IEEE Communications Society, Piscataway, NJ, pp 110–120
67. Little TDC, Ghafoor A, Chen CYR, Chang CS, Berra PB (1991) Multimedia synchronization. The Quarterly Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. 14:26–35
68. Mills DL (1991) Internet time synchronization: the network time protocol. IEEE Trans on Commun 39:1482–1493
69. Ramjee R, Kurose J, Towsley D, Schulzrinne H (1994) Adaptive playout mechanisms for packetized audio applications in wide-area networks. Proceedings of the Conference on Computer Communications (IEEE Infocom), Toronto, Canada, IEEE Computer Society Press, Los Alamitos, CA, pp 680–688
70. AT&T (1987) Streams programmers guide and streams primer. Englewood Cliffs, Prentice-Hall
71. McCane S, Jacobson V (1993) A BSD packet filter: a new architecture for user-level packet capture. Proceedings of the Usenix Winter Conference, San Diego, Calif., Usenix, pp 259–269
72. Pasieka M, Crumley P, Marks A, Infortuna A (1991) Distributed multimedia: how can the necessary data rates be supported? Proceedings of the Usenix Summer Conference, Nashville, Tenn., Usenix, pp 169–182
73. Clark D, Jacobson V, Romkey J, Salwen M (1989) An analysis of TCP processing overhead. IEEE Communications Magazine 27:23–29
74. Wolman A, Voelker G, Thekkath CA (1994) Latency analysis of TCP on A TM network. Proceedings of the Usenix Winter Conference, San Francisco, Calif., Usenix, pp 167–179
75. Partridge C, Pink S (1993) A faster UDP, IEEE/ACM Trans Networking 1:429–440
76. Topolcic C (1990) Experimental internet stream protocol, version 2 (ST-II). Request for Comments (Experimental) RFC 1190, Internet Engineering Task Force, (Obsoleted by RFC1819)
77. Delgrossi L, Herrtwich RG, Hoffmann FO (1994) An implementation of ST-II for the Heidelberg transport system. Internetworking: Research and Experience, 5:43–69
78. Gaver WW (1991) Sound support for collaboration. In: Bannon L, Robinson M, Schmidt K (eds) Proceedings of the 2nd European Conference on Computer-Supported Cooperative Work (ECSCW'91) Amsterdam, The Netherlands, Kluwer, Amsterdam, pp 293–308
79. Chinn HA, Gannett DK, Morris RM (1940) A new standard volume indicator and reference level. Bell System Technical Journal, 19:94–137
80. Angebranndt S, Hyde RL, Luong DH, Siravara N, Schmandt C (1991) Integrating audio and telephony in a distributed workstation environment. Proceedings of the Usenix Summer Conference, Nashville, Tenn., Usenix, pp 419–435
81. Neville-Neil GV (1992) Current efforts in client/server audio. X Resource, 8:69–86
82. Fulton J, Renda G (1994) The network audio system. X Technical Conference, X Consortium, Cambridge, MA; The X Resource is published by O'Reilly, Sebastapol, CA. Also X Resource, Issue 9, p 181–194
83. Levergood TM, Payne AC, Gettys J, Treese WG, Stewart LC (1993) Audiofile: a network-transparent system for distributed audio applications. Proceedings of the Usenix Summer Conference, Cincinnati, Ohio, Usenix, pp 219–236
84. Levergood TM, Payne AC, James G, Treese WG, Stewart LC (1993) AudioFile: A network-transparent system for distributed audio applications. Technical Report 93/8, Digital Equipment Corporation, Cambridge Research Lab, Cambridge, Mass
85. Schnorf P (1993) Integrating video into an application framework. Proceedings of ACM Multimedia, Anaheim, Calif., ACM, pp 411–417
86. Sun Microsystems (1994) XIL Programmer's Guide. Sun Microsystems, Mountain View, Calif
87. Rennison E, Baker R, Kim DD, Lim Y-H (1992) MuX: an X coexistent time-based multimedia I/O server. The X Resource 1:213–233
88. Zimmermann C (1994) Making distributed multimedia systems secure: the switchboard approach. ACM Operating Syst Rev 28:88–100
89. Center CC (1993) Cert advisory ca-93:15: /usr/lib/sendmail, /bin/tar and /dev/audio vulnerabilities
90. Tang JC, Isaacs EA (1992) Why do users like video? studies of multimedia-supported collaboration. Technical Report TR-92-5, Sun Microsystems Laboratories, Mountain View, Calif
91. Reddy ALN, Wyllie J (1993) Disk scheduling in a multimedia I/O system. Proceedings of ACM Multimedia, Anaheim, Calif., ACM, pp 225–233
92. Yu C, Sun W, Bitton D, Yang Q, Bruno R, Yus J (1989) Efficient placement of audio data on optical disks for real-time applications. Commun ACM, 7:862–871
93. Ramakrishnan K, Vaitzblit L, Gray C, Vahalia U, Ting D, Tzelnic P, Glaser S, Duso W (1993) Opertaing system support for a video-on-demand file service. Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, U.K., Lecture Notes in Computer Science 846, pp 225–236

HENNING SCHULZRINNE received his undergraduate degree in economics and electrical engineering from the Technische Hochschule in Darmstadt, Germany, in 1984, his MSEE degree as a Fulbright scholar from the University of Cincinnati, Ohio, and his Ph.D. degree from the University of Massachusetts in Amherst, Massachusetts in 1987 and 1992, respectively. From 1992 to 1994, he was a member of technical staff at AT&T Bell Laboratories, Murray Hill. In 1994, he joined GMD-Fokus (Berlin) as a postdoctoral researcher. His research interests encompass real-time network services, the Internet and modeling and performance evaluation.