GMD Research Series

GMD –
Forschungszentrum
Informationstechnik
GmbH

Changpeng Fan

# Supporting Continuous Media Communications and Applications by Exploiting their Special Characteristics

N° 4/1998

GMD –
Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de

**Anschrift des Verfassers/Address of the author:**
Changpeng Fan
Institut für Offene Kommunikationssysteme
GMD – Forschungszentrum Informationstechnik GmbH
Kaiserin-Augusta-Allee 31
D-10589 Berlin-Charlottenburg
E-mail: Changpeng.Fan@gmd.de

D 83

**Die Deutsche Bibliothek - CIP-Einheitsaufnahme**

**Fan, Changpeng:**
Supporting continuous media communications and applications by exploiting their special characteristics / Changpeng Fan.
GMD-Forschungszentrum Informationstechnik GmbH. - Sankt Augustin : GMD-Forschungszentrum Informationstechnik, 1998
(GMD research series ; 1998, No. 4)
Zugl.: Berlin, Techn. Univ., Diss., 1998
ISBN 3-88457-328-4

**ISSN 1435-2699**
**ISBN 3-88457-328-4**

# Abstract

Distributed multimedia applications need end-to-end support from both the networks and the endsystems. The operating systems are critical both for accessing networking services and for controlling the activities on the endsystems. It is therefore vital to enhance the functionality and performance of the operating system in order to provide feasible support for multimedia communications and applications. The work proposes to support multimedia, especially continuous media, communications and applications by exploiting their special characteristics. The main emphasis is on OS support for the real-time aspect of continuous media.

The work proposes to realize a soft real-time framework for the operating system with supporting features for multimedia. All the system activities are controlled by the framework so that predictability can be achieved. A set of feasible scheduling methods for the framework have been designed and evaluated. As an essential component for the soft real-time system, various soft real-time handling methods for timing overflow are used. In addition, predictable protocol processing architectures are explored and a flexible and adaptive service provision model is proposed to support the macro-level adaptability of multimedia applications.

The models and algorithms proposed in this work have been validated through theoretical analyses, simulation evaluations or experimental implementations. Together, they can be used to construct multimedia supporting systems which are predictable, flexible and efficient.

# Zusammenfassung

Die vorliegende Arbeit befaßt sich mit der Unterstützung verteilter multimedialer Anwendungen durch das Betriebssystem.

Aus Echtzeit- und Performancegründen benötigen verteilte multimediale Anwendungen besondere Unterstützung sowohl durch das Netzwerk als auch durch das Betriebssystem. Das Betriebssystem spielt dabei eine kritische Rolle. Es ermöglicht multimedialen Anwendungen, auf Netzwerkdienste zuzugreifen und verwaltet gleichzeitig alle Systemvorgänge auf dem Endsystem. Zur optimalen Unterstützung verteilter multimedialer Anwendungen ist es erforderlich, das Betriebssystem so zu verbessern, daß es geeignete leistungsfähige Dienste bereitstellt. Dies kann insbesondere durch die Ausnutzung der besonderen Eigenschaften multimedialer Anwendungen erreicht werden.

Um den spezifischen Zeitanforderungen multimedialer Anwendungen gerecht zu werden, wird ein “Soft Real-Time Framework” als Systemkonzept für ein multimedia-unterstützendes Betriebssystem vorgeschlagen. Gegenstand des Frameworks sind der Entwurf und die Bewertung von Echtzeitplanungs- und Echtzeitbehandlungsmechanismen, die den besonderen zeitlichen Anforderungen von multimedialen Anwendungen Rechnung tragen. Die sich in diesem Zusammenhang ergebenden besonderen Anforderungen an Protokollbearbeitung und adaptives Dienstanbieten (adaptive service provision) werden in dieser Arbeit ebenfalls berücksichtigt.

Mittels theoretischer Analyse, Simulation und Implementierung werden die in der Arbeit beschriebenen Modelle und Algorithmen validiert und evaluiert. Es wird gezeigt, daß sie für den Aufbau von effizienten, flexiblen und multimediafähigen Unterstützungssystemen geeignet sind.

To

my parents.

# Acknowledgments

First of all, I would like to acknowledge the encouragement and guidance of Prof. Dr. Radu Popescu-Zeletin, without whom this work would not have been possible.

I would also like to thank Prof. Dr. Adam Wolisz for the encouragement and the comments on the work.

Most of my work is conducted in the excellent academic environment of GMD FOKUS. My thanks to all my former and current colleagues at FOKUS, especially Drs. Thomas Luckenbach, Markus Walch and Rainer Schatzmayr, for the helpful discussions. Thanks also to Mr. Jürgen Selent for the assistance in many ways.

Finally, I wish to express my gratitude to all members of my family for their encouragement, understanding, love and support.

# Table of Contents

## PART II Using a Soft Real-Time Framework

# List of Figures

# List of Tables

# PART I

# Introduction and Background

# Chapter 1

# Introduction

Distributed multimedia system has become one of the main focuses of computer networking and applications. The technical advances in two fields have made a distributed multimedia system feasible. On the one hand, the rapid advances in hardware and software have brought about a new generation of workstations and personal computers with low-cost audio and video capabilities. On the other hand, with the technical advances in fiber-optic and VLSI, broadband networks can now provide a larger bandwidth than ever before. We have already seen the basic audio/video capabilities on our desktops, which have made some simple desktop teleconferencing possible: audio is now common in workstations and high-end PCs; though not as common, video devices are coming very rapidly; workstations are fast enough to do basic software compressions and decompressions, which makes software video display possible without the use of specialized video hardware; and many networks are fast enough to carry several compressed A/V streams.

The trend of current development is to extend and optimize the multimedia applications on endsystems (workstations, PCs), and to further advance the distributed multimedia applications by connecting the endsystems with large-bandwidth, high-speed networks.

In the presented work, we investigate a variety of issues related to operating system support for multimedia communications and applications. The main effort is to develop strategies, models and algorithms for supporting continuous media by exploiting their special characteristics, which are not present in or are not typical of other kinds of computer applications.

In this chapter, the motivations and goals of our work are sketched. The organization of the dissertation is also briefly explained.

## 1.1 Motivation and goals

### 1.1.1 Motivation

Multimedia (MM) systems involve the processing of continuous media (CM) such as live digital audio / video and the traditional discrete media such as text files. Multimedia applications have posed a set of new functional and performance requirements on hardware and software components of a computer system. In a distributed / networked multimedia system, a single integrated network such as a B-ISDN network based on ATM technology will carry a mixed traffic of a variety of media (continuous as well as discrete media) and the endsystems (host computers) will process the continuous media much in the same way as discrete media.

- **Networked / distributed multimedia systems need networking as well as endsystem support**

For networked / distributed multimedia systems, both the support from the network and the support from the endsystem are needed. Until recently, much emphasis has been put on the networking aspect in order to construct a multiservice network. The provision of quality-of-service (QoS) guarantee has been extensively investigated in the networking field. Such network QoS parameters as bandwidth, delay and error rate are supported now by ATM technology on a per connection (or in the term of ATM, a *virtual circuit*) basis. The same issues in internetworking environments are also well investigated as can be found in the IETF work on building an Integrated Services Packet Network [Clark92, Shenker95a, Braden96] and the work by BERKOM network [Zeletin89] or Berkeley Tenet group [Ferrari92].

In contrast, relatively less attention seemed to have been paid to end-systems and the operating system which controls the activities on the end-systems. We note that in multimedia systems, it is not sufficient to guarantee the transmission of MM information flow between two endsystems at the level of networking service. In the end, it is the endsystem (controlled by an OS) that uses this networking service to provide multimedia service to end-users. Enhancements in both the endsystem hardware and software are needed in order to meet the time-constrained high processing requirements of MM applications.

Traditionally, the network is usually the bottleneck of the system scenario. But some recent experiments with high-speed network such as Xunet have shown that the endsystem can easily become the bottleneck where the network bandwidth is plentiful [McCanne95].

- **Operating system support is crucial for both the endsystems and the networks**

Traditionally, the operating system plans a critical role on the endsystem. It provides system services to all other software components by controlling effectively all the hardware resources. It is also the task of the operating system to coordinate all the hardware and software activities.

Support for continuous media communication is one of the key challenges in constructing a distributed MM system. The QoS requirements of the applications in regard to communication should be converted to QoS parameters supported by the network. It is the operating system that controls the efficient and predictive access of the networking interfaces and thus the networking services (see Figure 1-1).

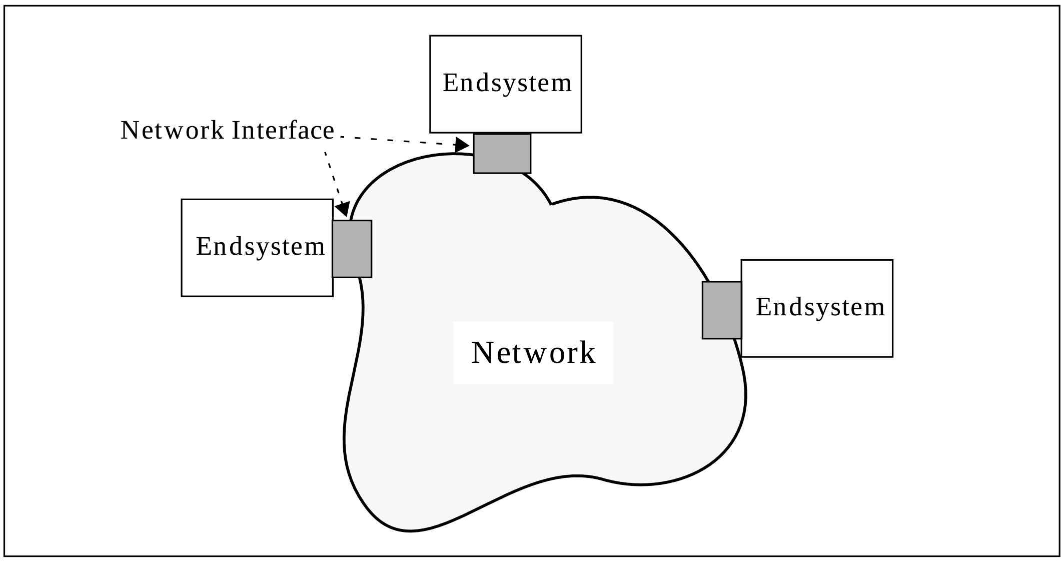

***Figure 1-1*** **Endsystems with network interfaces to network**

In order to guarantee the real time and high-bandwidth transmission of MM information flow between two MM applications on two end-systems, it is not sufficient to guarantee the real time transmission of MM data flow between two end-points up to network interfaces or even up to the level of transport service (protocol processing). It is also important that the operating system provides appropriate response for all other processes that use these MM information. Hence, QoS issues such as throughput, latency and real-time responsiveness are end-system concerns as well as concerns of the communication infrastructure. Some transport services such as the services with real time features can only be feasibly implemented and provided within a suitable OS supporting environment. The whole solution requires an integration of the operating system and the communication infrastructure.

- **Currently available OS supports are not feasible / not enough for MM**

As stated, multimedia applications have posed a set of new functional and performance requirements on hardware and software components of a computer system. Especially, continuous media (CM) represents a significant departure from traditional applications and places a new set of constraints on the operating system supporting services. Two constraints among them are dominant requirements on the operating systems which try to support multimedia, i.e., the timeliness required to simulate continuous media (the continuous media are perceived to change over time) and the unavailability of brute force resources with which to do so (for example, the memory resource required to “play” these media exceed the memory available on many workstations). Especially, the timing requirements and the guaranteed throughput required for continuous media force specific consideration in designing operating systems aimed at CM support. Communication applications running under conventional operating systems like UNIX are at the mercy of unpredictable delay and jitter caused by page swaps, interrupts, multiplexed protocol stacks and other application activities on the same systems. Figure 1-2 shows briefly the system activities on a BSD UNIX system or its derivative like SunOS 4.1.*. In practice, these factors make conventional OS unsuitable for anything but the least demanding ones of distributed multimedia applications. Neither have the commercially available real-time operating systems provided the feasible services for CM. Enhancements to OS in both functionality and performance are desired in order to provide feasible support for multimedia applications.

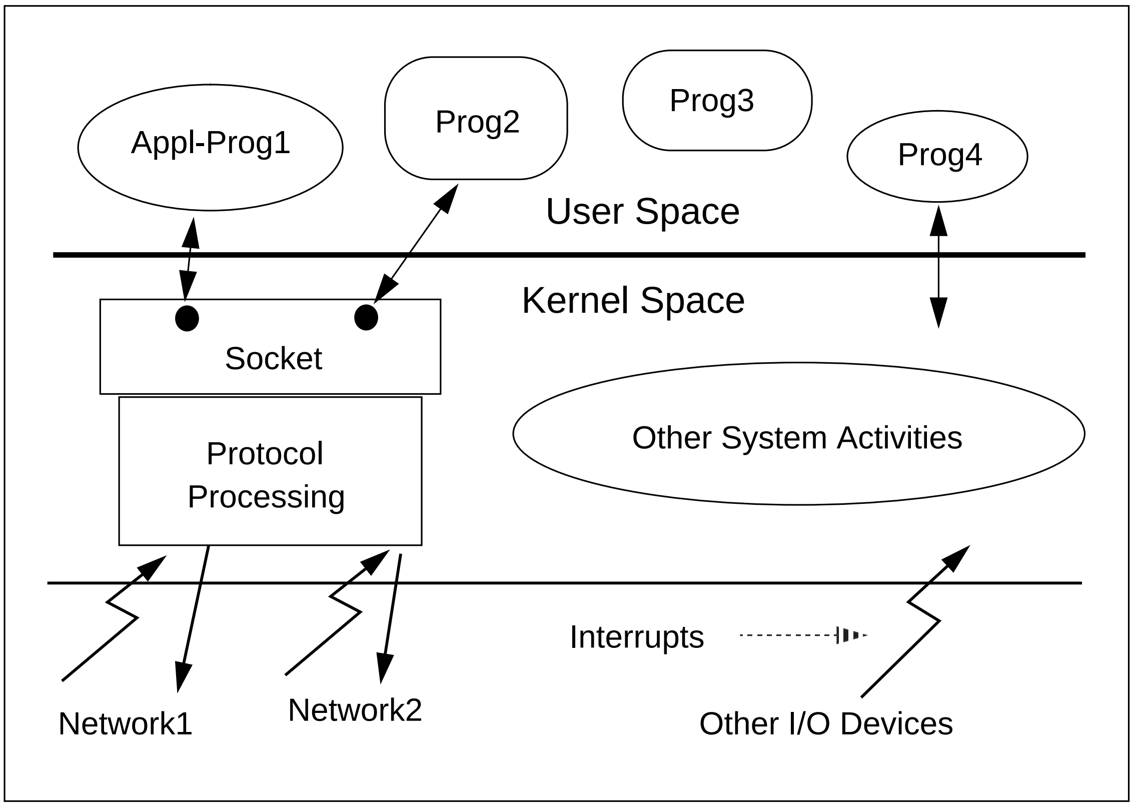

***Figure 1-2*** **System Activities in BSD UNIX**

- **Summary**

In summary, the main motivation of our work is:

The distributed multimedia applications need end-to-end performance support from both networks and endsystems; the operating systems are critical both in accessing networking services and in controlling the activities on the endsystems; and we want to make contributions to enhance the functionality and performance of the operating systems in order to provide feasible supports for multimedia communications and applications, especially for the real-time aspect of such supports.

## 1.1.2 Application and problem domain

Since there are a large variety of multimedia application possibilities and the resulting issues of the OS support for multimedia are numerous, we try to identify the main application and problem domain of our interest in this section. The main OS supporting issues, on which we will concentrate, will be given in the next section.

- **Support of continuous media is challenging in two aspects**

As mentioned above, continuous media (CM) support is challenging in two aspects: the timeliness required to simulate continuous media (the continuous media are perceived to change over time), and the high performance and the huge resource needed to do so (for example, the bandwidth and memory requirements on the network and the workstations). As an example, the uncompressed bit rates for digital video rages from 44 Mb/s (NTSC video quality) to 1.5 Gb/s (HDTV video quality). And the video should be regularly delivered with a rate of 30 frames/second in real time. Audio, in contrast, should be packetized every 20ms to 100ms with a constant bit rate.

***Table 1-1*** **Some characteristics of digital audio and video**

| Characteristics | Audio | Video |
|---|---|---|
| Continuous timing | required | required |
| Rate | 13 ... 64 ... 1500 ... kb/s | 200 ... 1500 ... 6000 ... kb/s |
| Traffic | constant + silences | variable bit rate |
| Packet size | small | large |
| Loss tolerance | usually <= 5% | $10^{-5}$ ... 15% |
| One-way delay tolerance: conference | 40 ms (without echo cancellation) ... 150 ms (echo cancellation) | up to 200 - 300 ms |
| Playback delay tolerance | >= 500 ms | >= 500 ms |

Through compression and decompression, possibly aided by special hardware, all of the problems concerning high data rate, large data volume and stringent timeliness can be alleviated by one or two orders of magnitude. Still, these requirements are quite significant for the capabilities of current workstations and high-end PCs.

Therefore, it is critical for the supporting operating system to try to “preserve” the timeliness and bandwidth of the low-layer hardware and software components up to the use of high-layer application components, and to coordinate all the system components and activities in a harmonic way.

- **Multimedia collaboration (MMC) as a typical multimedia application**

The scenarios of MM applications are diverse. We will mainly pay our attention to a MM application domain called multimedia collaboration (MMC). Such applications have found their practical usage in many circumstances and are implemented in many projects such as the BERKOM-II MMC system [Altenhofen93]. They are the good technical bases for realizing Computer Supported Cooperative Work (CSCW).

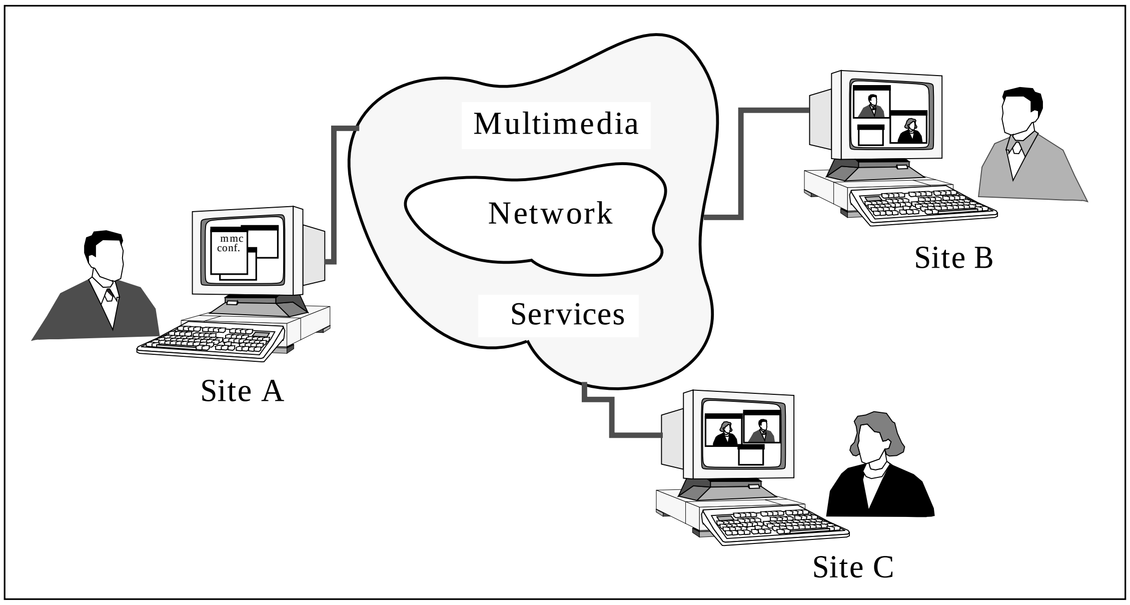

***Figure 1-3*** **MMC scenario with 3 sites**

In a typical MMC scenario, several users collaborate their work by working on some commonly accessible documents concurrently, and they usually interact with one another by way of live interactive audio/video connections. Virtually, users (participants) see and hear each other in a natural way as in a face-to-face situation.

In such a scenario, an end-system should handle continuous media communication and processing (live audio / video), discrete media communication and processing (document exchange and processing), as well as other local processing (mail processing, background compilation, etc.). The two requirements of this scenario are: the users want to have the real-time continuous media processing capability and at the same time, the users want to have other conventional processing capability. Figure 1-3 shows a MMC scenario with 3 sites, where site B and site C posses full audio / video capabilities and site A has only audio capability.

Note the key point here is that ***both*** requirements should be satisfied at the same time, because the parallel availability of the continuous media and the discrete media is necessary for the users' collaboration. If, for example, the document (a discrete media) can not be exchanged reasonably fast because the real-time interactive audio / video transmission (a continuous media) have used up all the processing and networking resources then the users can not conduct a satisfactory collaboration. The same undesired effect can happen for the other way round: if the users have only easy access to some common documents but the quality of live interactive A / V transmission is bad, then the interactive collaboration can not be satisfied either.

Exactly here lies the main challenge. On the one hand, the capacity of a typical, currently available workstation is powerful enough to support live digital audio / video, if the processing and networking capabilities are carefully allocated, scheduled and used. On the other hand, this capacity is not yet enough to support multiple activities of multimedia and non-multimedia at the same time without a very careful arrangement. The designer for a supporting operating system for multimedia should always keep this in mind.

- **Multimedia supporting affects every part of the multimedia operating system**

There are four major components in a conventional operating system [Tanenbaum87]: process management, input / output (device drivers), memory management, and file system. Since MM applications will usually need to coexist with traditional applications such as data crunching and word processing, the optimization of the functions of a multimedia operating system (MMOS) should take this coexistence into consideration. As can be seen clearly through a brief survey of the current literature and ongoing work, the need to accommodate continuous media delivery and processing is already affecting every part of the operating system: process and thread scheduling, network access control and communication protocol, memory allocation and file system. A number of key technologies are beginning to emerge and one of the most important task is to extract the general principles of them and to integrate them to produce complete solutions.

- **Summary**

To summarize, the multimedia operating system should cope with the challenges of continuous media in terms of stringent timeliness, high data rate and huge data volume. We have to meet the requirements of accommodating both continuous media and discrete media in one system as is necessary for the multimedia collaboration application MMC. And we have to develop general principles which can be integrated into all the major components of a multimedia operating system.

### 1.1.3 Goals

Numerous issues should be addressed in order to provide feasible OS support for multimedia. The following will try to identify the main issues on which we will concentrate. The basic ideas to deal with these issues are also given as part of the formulation.

As our goals, we want to tackle several important OS supporting issues and propose appropriate solutions to them so that these proposals can be integrated into a multimedia-feasible operating system.

- **To formulate a new set of principles on resource management and scheduling**

First of all, operating systems that support multimedia applications must modify the traditional view of scheduling and resource allocation in order to accommodate the need for timeliness and to deal with the constraint of limited resource. It is very important to formulate a new set of basic principles on resource management and scheduling which can, first of all, accommodate the requirements of continuous media and which can be applied to every part and every aspect of the multimedia operating system.

In our opinion, the central idea and starting point for supporting CM communication and applications should be to exploit the special features of them. That is, the support of OS to continuous media should exploit the special features of continuous media which are not present in or are not typical of other kinds of computer and communication applications. Some of these features can be identified as soft real-time, soft (not 100%) guarantee, periodicity, error-tolerance, adaptability, etc.

Our new view on resource management and scheduling will also consider the need to accommodate the continuous media as well as the conventional discrete media at the same time in order to meet the requirements of such applications as MMC.

- **To develop a soft real-time framework for the whole multimedia system**

It is clear that multimedia operating systems must have real-time features and other supporting features for multimedia applications. We would like to point out here that some rudimentary proposals can not tackle the whole problem of real-time MM support. For example, only giving the highest priority to communication protocol processing can only achieve the effect that the multimedia data packets are processed as soon as possible. This is, however, not enough for the whole system effect, since the application processes that consume these MM data can not be scheduled to use these data in time and according to their periodicities.

We hold the view that real-time scheduling schemes should be applied to all the system components. We note that MM applications are not hard real-time applications in themselves. A 100% guarantee is not necessary in most cases. We will investigate the applicabilities of the traditional real-time scheduling theory to multimedia systems. We consider the realization of a soft real-time framework for the whole multimedia system as an appropriate solution.

- **To develop and evaluate soft real-time scheduling and handling schemes for multimedia**

Feasible soft real-time scheduling schemes are the cornerstones for the functionality and efficiency of the soft real-time framework. The basic requirements on the base scheduling methods are that they should be simple, flexible yet powerful. Our work will emphasize on both predictability and flexibility. Base scheduling methods will be designed to reflect the cooperative timing enforcement model as well as the semi-imperative timing enforcement model.

Our soft real-time framework should show its “softness” in mainly two aspects: first, different guarantee degrees should be supported; second, the timing specifications of processes should be flexible and the timing exceptions should be handled in some “soft” and “graceful” manners. The second aspect is necessary, since we do not require a rigid timing specification from the MM processes and not-so-often timing violations both from the side of user and from the side of the system are allowed. For this purpose, different soft real-time handling methods for timing overflow will be developed.

- **To build a feasible protocol processing architecture**

Protocol processing subsystem plays an important role in constructing an efficient and predictable distributed multimedia systems.

We will describe the schemes to structure the protocol processing subsystem in such a way that the whole subsystem runs under the control of the soft real-time scheduling schemes so that the protocol processing is also predictable.

In addition, there are several important issues which are critical for the efficiency and the predictability of the protocol processing subsystem. We will investigate the possibility of damping the effects of interrupts and the different methods for providing communication services to communication users.

- **To provide adaptive QoS support for applications**

The services of applications affects the end-users most directly. Quality of Service (QoS) is a central concept in supporting MM applications. From the viewpoint of the supporting operating systems, the requirements of the upper layer CM applications can usually be expressed in the form of a set of QoS parameters. According to strongness and applicability, a category of different degrees of QoS support can be identified: from best-effort QoS support, soft guarantee of QoS, to relatively hard guarantee of QoS. In addition, the QoS requirements of an application can also change during its life-time by switching between several “working modes”. We believe that a range degrees of QoS support are needed by different application types and that a flexible and adaptive support should be provided. Actually, one typical nature of many multimedia applications lies in the fact that they are not rigid but adaptive, not intolerant but tolerant. In the range of its tolerance, the variation of QoS provision is regarded by an application as acceptable.

At the lower level, a set of micro-level OS mechanisms should be provided to implement a processing and transport system that is predictable enough to make various degrees of guarantees possible. These have been found in the above in the form of soft real-time scheduling and handling mechanisms as well as predictable protocol processing structures.

At a higher macro-level, many multimedia communication applications are flexible and adaptive so that they can tolerate the changes in their environmental parameters to some extent. The processing and transport system should exploit this feature and support this feature explicitly. We propose to use a **f**lexible and **a**daptive **s**ervice suppor**t**ing model (FAST) to deal with both functionality and performance requirements. The FAST model will deal with both system-initiated adaptation and user-initiated adaptation.

- **Summary**

As our goals, we will formulate a new set of principles on resource management and scheduling, we will develop a predictable and flexible soft real-time framework for the whole multimedia system and we will develop and evaluate soft real-time scheduling and handling schemes for multimedia. Taking the applicability of the soft real-time scheduling schemes into account, we will develop feasible protocol processing architectures that are efficient and predictable. In order to provide adaptive QoS support for CM communications and applications, an adaptive service model will be used for constructing the system.

## 1.2 Organization of the dissertation

The dissertation consists of 10 chapters and is divided into 5 parts.

PART I Introduction and Background:

After motivating our work and setting our goals in this chapter, we overview the general issues related to OS support for continuous media in Chapter 2. A brief state-of-the-art survey on major areas of operating system support for multimedia is presented. We then give the system models which are used throughout our work. The basic approaches of our research are then outlined. This includes a presentation of our basic views on resource management and scheduling.

PART II Using a Soft Real-Time Framework:

Chapter 3 deals with the relationship between real time scheduling and continuous media communications and applications and proposes a soft real-time framework as a solution to real-time continuous media support. We argue that CM applications and communications posses a set of “soft” features which can be accommodated in our real-time process model and scheduling framework.

Chapter 4 continues our arguments by looking at some closely-related issues. Soft real-time framework, scheduling framework and soft real-time process model are concretized. An analysis of the suitability of different real-time scheduling methods is also conducted. The chapter also explores the mathematical bounds of soft real-time with loss by using analyses based on the queueing theory.

PART III Soft Real-Time Scheduling and Handling:

The first chapter of this part proposes and justifies some soft real-time scheduling and handling schemes which can be used in our soft real-time framework. The scheduling schemes are based on cooperative model, semi-imperative model and elastic-time model respectively. The handling schemes can be chosen to deal with different situation of timing overflow.

Both simulation and implementation are used to evaluate our soft real-time scheduling and handling schemes. Numerous qualitative and quantitative evaluations are presented in both chapters. Chapter 6 presents the soft real-time effects by simulations on both simple theoretical scenarios and composite practical scenarios. Chapter 7 gives details to some of our experimental implementations. Measurement results from the implementation are used to further validate our models and algorithms.

PART IV  Complementary Techniques:

Chapter 8 follows by considering other problems in addition to scheduling in order to realize a highly-efficient and predictable protocol processing subsystem. We discuss how to damp the effect of interrupts, how to structure the protocol processing components and other implementation-essential issues.

Many of the continuous media applications are adaptive in nature and there exist such technical bases as media scaling and transport protocol layer QoS renegotiation. Based on the techniques developed so far which guarantee a certain level of predictability, the adaptive service model presented in Chapter 9 integrates these elements together to best support the implementation of adaptive continuous media applications. Corresponding support from scheduler and other resources pools are discussed.

PART V  Conclusion:

As a conclusion, Chapter 10 summarizes our main contributions and compares them with other related current research and development. We also point out the possible directions for future research.

# Chapter 2

# An overview of OS support for continuous media and an outline of our system models

In order to get a general understanding of the necessity to provide multimedia-specific operating system support and of the main areas of current investigation, we make a brief survey on the current practices of OS support for multimedia. We then provide our models about end-systems and networks to clarify the usability scope of our proposed strategies and algorithms. The basic approaches of our work are also presented as a basis for further elaboration of our work.

Although the term multimedia can be used to refer to discrete media such as texts, graphics and still images in its broader sense, we are mainly concerned with continuous media such as digital audio and video. It is technically more challenging to support continuous media, which are continuous information flows over real time. In our context, multimedia refers mostly to continuous media.

## 2.1 A brief state-of-the-art survey

A conventional operating system consists of mainly four components [Tanenbaum87]: process management, input/output (device drivers), memory management, and file system. Multimedia applications will usually need to coexist with traditional applications such as data crunching and word processing, the optimization of the functions of real-time multimedia OS should take this coexistence into consideration. A brief survey of the current literature will show that the need to accommodate continuous media delivery and processing is already affecting every part of the operating system: process and thread scheduling, network access control and communication protocol, memory allocation and file system. The following subsections identify and survey some of the areas under current investigation. Because it is necessary to understand the applications' behavior and needs in order to provide the feasible support for them, a brief analysis of the implementation schemes of some typical MM applications is also conducted.

An analysis on different degrees of media integrations and their significances on OS can be found in *Section 2.2.1 Integrated MM end-system and software-intensive approach*. Further references to related work are given throughout the whole dissertation, whenever necessary. And, we will highlight again the differences between our work and related work in *Section 10.1 Summary of main contributions and comparisons with related work.*

### 2.1.1 OS kernel support for MM

The existing popular operating systems such as UNIX and its derivatives are inherently unsuitable for real-time continuous multimedia such as digital audio and video. These operating systems have been designed to maximize the performance of compute-intensive applications and to maintain a fair share of resources among competing users. The techniques used by them are sometimes infeasible or even counter-productive for MM applications.

There are several research efforts trying to extend the OS kernel functionality to support continuous MM.

There are mainly two approaches to this. One is represented by the work like "real-time UNIX" [Fisher92, Hagsand94]. The work by Tenet group [Fisher92, Vetter92] has shown that real-time capabilities can be wedged into existing UNIX-like OS by placing preemption points into the kernel to provide a certain real-time responsiveness required by continuous MM. The main concern with this approach is that it does not explicitly tackle some real-time issues such as priority inversion and it usually demands conservative and excessive reservation in order to provide a certain guarantee. Another approach detailed by examples below begins by directly addressing the real-time requirements by MM applications, including the standard real-time problems of priority, deadline scheduling, and priority inversion.

Many applications that use continuous media need guaranteed end-to-end performance (bounds on throughput and delay). To reliably support these requirements, systems components such as CPU schedulers, networks, and file systems must offer performance guarantee. The work by the DASH group at UC Berkeley [Anderson90b, Govindan91] proposes to use a meta-scheduler to coordinates these components, negotiating end-to-end guarantees on behalf of clients. They propose a CM-resource model as a basis for such a meta-scheduler. A split-level scheduling mechanism is used to implement a deadline-workahead scheduling to minimize system call overhead and to deal with timing requirements of MM applications.

The group led by K. Jeffay at UNC [Jeffay92] based its work on YARTOS kernel [Jeffay91] — an operating systems kernel that provides real-time communication and computation services. They aimed at workstations with dependent A/V subsystems. The effective use of these A/V subsystems requires that real-time services be provided at a number of level within the kernel.

Besides, the ARTS group at CMU [Tokuda92, Tokuda93] and a group at Fujitsu [Nakajima91, Nakajima92] have made extensions to the Mach OS to include real-time and multimedia capabilities. In their SUMO project, the Distributed Multimedia Research Group at Lancaster University is extending CHORUS OS to support MM communications [Blair92b, Coulson93, Coulson93a]. They used the flow concept to represent the complete path of a media stream from source to destination.

A more recent research collaborate project Pegasus is being conducted by the University of Twente and the University of Cambridge [Leslie93, Mullender94, Sijben95]. They are designing and implementing a microkernel called Nemesis for the multimedia workstations and the Pegasus file system. The kernel uses a single address space shared by multiple protection domains and the scheduling mechanism implemented in the kernel is similar to the scheduler activations mechanism.

### 2.1.2 File system and file storage server for MM

A file system for MM must support solutions to the problems of synchronization, timeliness and bandwidth in order to meet the demands of MM applications. Current workstation file systems lack this capability. The earlier work by the DASH group showed that the efficient and effective access to even simple MM data requires the use of both new layout policies and deadline based scheduling of file access. The real-time multimedia OS file system must support a bounded model of read-ahead directly into the application data space. It can allow neither the exhaustion of data nor the overflow of the available buffer space.

At UCSD, Rangan and others [Rangan91] describe both algorithms and limits on buffer resources necessary to satisfy these requirements under varying constraints. Some techniques for physical organizations of multimedia storage like “disk striping” have also been proposed [Adam94]. Other systems, e.g. the Lancaster Media Storage Server [Lougher94], use RAID technology to achieve the high bandwidth and storage capacity needed for continuous media.

### 2.1.3 Network access and communication protocols

Since network is a shared resource, network access is also a concern in supporting MM applications. The Quality of Service (QoS) parameters [Danthine92] required by MM communications at the high levels will partly be mapped onto and implemented by the QoS parameters at the networking low levels. The new networking technologies such as ATM provide some kinds of reservations and guarantees at the networking level, making the reservation and guarantee of communication QoS parameters at above layers possible.

Many communication protocols have also also designed and implemented which aim to address the special requirements of continuous media communications. Examples can be found in the work by the Tenet group at ICSI [Ferrari92, Wolfinger92], the ongoing work by the Internet IETF audio/video transmission group [Schulzrinne92]. While the protocols such as Rapid [Schatzmayr94, Schatzmayr96] used Forward Error Correction mechanisms to reduce delay and buffer burden in case of error, there is also work which shows the feasibility of retransmission in a controlled manner [Papadopoulos96]. It should be noted that the implementation and the utilization of these protocols would not be complete or even possible without the suitable supports from operating systems.

### 2.1.4 Implementation schemes of typical applications

Without loosing generality, we classify common multimedia applications into the following three categories by identifying their main timing requirements:

Type 1 — Real-time interactive. These MM applications have the most stringent timing requirements. The applications usually try to keep a real-time, continuous information flow. And the information exchange is quite often bidirectional and interactive. Many desktop video conferencing tools [Hewitt96] belong to this category. In order to achieve the effect of interactive real-timeliness, the end-to-end delay of the information flow should usually be strictly bounded. Delay variations (jitters) should also be kept small.

Type 2 — Continuous replay. The timing requirements of the applications in this category are less stringent than those in Type 1. Although the applications should still maintain a continuous information flow, the flow is quite often unidirectional and the end-to-end timing bound on the information replay is usually flexibler and larger than those of type 1. Video-on-demand applications belong to this category [Rangan91]. Note that the delays and delay jitters in these applications should also be controlled in a certain range, otherwise other problems might appear. For example, larger delay variations usually lead to larger buffer to compensate them.

Type 3 — Fetch-and-play. Although involved with audiovisual activity, the applications in this category belong more or less to the traditional data communication and processing applications. The information is usually fully fetched from the other side before it is consumed. The current World-Wide-Web (WWW) video components usually work in this way — the video source files are fetched completely from other servers before they are played at the local side.

The audio/video components of the Multimedia Collaboration (MMC) applications generally belong to application type 1 [Dittrich95]. In some special scenario such as unidirectional multicast transfer, they can also appear in the form of application type 2. The typical MBONE audio/video tools [Jacobson95, Schulzrinne95b, McCanne95] can be considered as type 1 applications when they are used in an interactive scenario. They sometimes also appear in the form of application type 2 as in the example of MBone VAT Radio Broadcast where the playback delay has been observed to range from 300ms to several seconds.

In the following, we are only interested in applications of type 1 and type 2, where some degrees of real-timeliness are involved. Type 1 applications are of special interests because they are technically more challenging. Both types of the applications can be abstracted as “playback applications” with different timing stringentness/rigidness. This will be dealt with in *Section 3.2.1 Modeling CM activity as playback application*.

## 2.2 System models and their justifications

After a brief state-of-the-art survey, we are now ready to present and justify our models concerning the architectures and the system components of the multimedia systems. We will also present and justify our views on some fundamental issues such as resource management, the relationship between reservation and guarantee. In addition, the emphases and the basic approaches of our work will be outlined.

### 2.2.1 Integrated MM end-system and software-intensive approach

A decisive factor in a multimedia system is how and to what extent the media data is handled within the computer system. The design of a multimedia operating system is fundamentally determined by how the media data is handled within the computer system [Mullender94, Schulzrinne95a, Tennenhouse95].

Mainly two control manners have been used in the existing multimedia systems: a control-only approach or a fully integrated approach. In a control-only approach, the continuous media data generally does not touch the operating system or main memory, but rather uses a separate infrastructure. In an integrated approach, the continuous media data are under the direct control of the operating system and the CM data can be processed by the application programs.

Most existing media toolkits and environments address the problem encountered in a control-only system. The goal of such systems can be briefly defined as “delivery of audio/video data to desktop”, although they differ in their ways and degrees of digitizing and transmitting audio or video. For the case of video in some earlier systems such as [Milazzo91], the control-only approach simply connects analog video to a separate monitor or uses analog picture-in-picture techniques. Analog mixing makes it difficult to integrate video into the windowing system, so that the user perceives it as an attachment rather than an integral part of the system. There is also likely to be a rather low upper bound on the number of concurrent video windows.

Instead of analog mixing, some systems such as the Pandora system [Hopper90] 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. A more integrated approach has live video and workstation graphics share the same frame buffer, as is done for many workstation-based video boards such as a Parallax board for SUN workstations. They all allow mostly seamless video integration into the windowing system.

The fully-integrated systems are emerging recently. Pegasus system [McAuley93, Leslie93, Mullender94] partly offers the choice of either treating continuous media as data to be processed or simply switched from network device to display device. ViewStation project of MIT [Tennenhouse95] aims explicitly at “delivery of media data to application”, which means making media data accessible to and manipulatable by the application program. Note that an integrated approach implies also a software-intensive approach at the same time, since the media data can now be accessed and processed by software programs.

In a control-only multimedia system, the media data do not touch the CPU on its path through the system, therefore their operating system requirements are much relaxed. They impose only some control timing constraints on the operating system. In an integrated multimedia system, in contrast, the operating system must be able to fully support and control the multimedia-related activities. The OS requirements are thus much complicated. But an integrated multimedia system with the possibility of software-intensive processing has many advantages that can not be found in a control-only system:

- First of all, audio and video are on longer second-class media on which the only operations are capture, storage and rendering, but media that can be processed — analyzed, filtered, modified — just like text and still images. Applications can analyze their audio and video data input and take actions based upon the analysis. “AI-like” functions like image categorization will be now possible with CPU intervention.

- Second, the audiovisual data processing can easily coexist with other traditional data processing. The incremental cost of adding video to an existing workstation can be close to zero if only software decoding is desired, as is evident by the fast popularity of most MBONE applications. 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.
- Third, the benefits of letting CM data use standard system resources can only be exploited by the users if CM data can be handled in the same software framework as other data types. Such a framework in a distributed computer system provides not only the services of the operating system, but also those of the communication network, the window system and the programming toolkits.
- Fourth, the software-intensive approach brings naturally with itself flexibility, scalability and adaptability. With evolving standards in the area of network protocols and media compression, software-intensive approaches offer far more flexibility. Software decoding can easily display several windows of live video simultaneously (depending on workstation performance), while most hardware decoders only support a single output window. Software-intensive applications also adapt well to the resources made available by the platform on which they are executing and to the dynamic load by the mix of concurrently active applications.

Because of the above advantages, the work on integrated multimedia systems is regarded as the main stream of the current and future research and development. Therefore, our work on OS support aims at fully integrated multimedia end-systems. Our goal-OS will not only provide support in accessing networking services but also provide support for controlling all the activities on the endsystems [Fan94a, Fan95a].

### 2.2.2 QoS architecture and components

The notion of quality of service (QoS) originally emerged in communications fields such as ISO-OSI to describe certain technical characteristics of data transmission. In our context, we use the term QoS to refer to all general system characteristics that influence the perceived quality of an application at any level as is shown in Figure 2-1. Intuitively, quality of service is the specification of how "good" an offered service is. QoS is generally characterized by a set of specific parameters. For our purpose, the following is used as a working definition [Steinmetz95, Vogel95]:

Quality of service (QoS) represents the set of those qualitative and quantitative characteristics of a distributed multimedia system which are necessary for the achievement of the required level of functionality of an application.

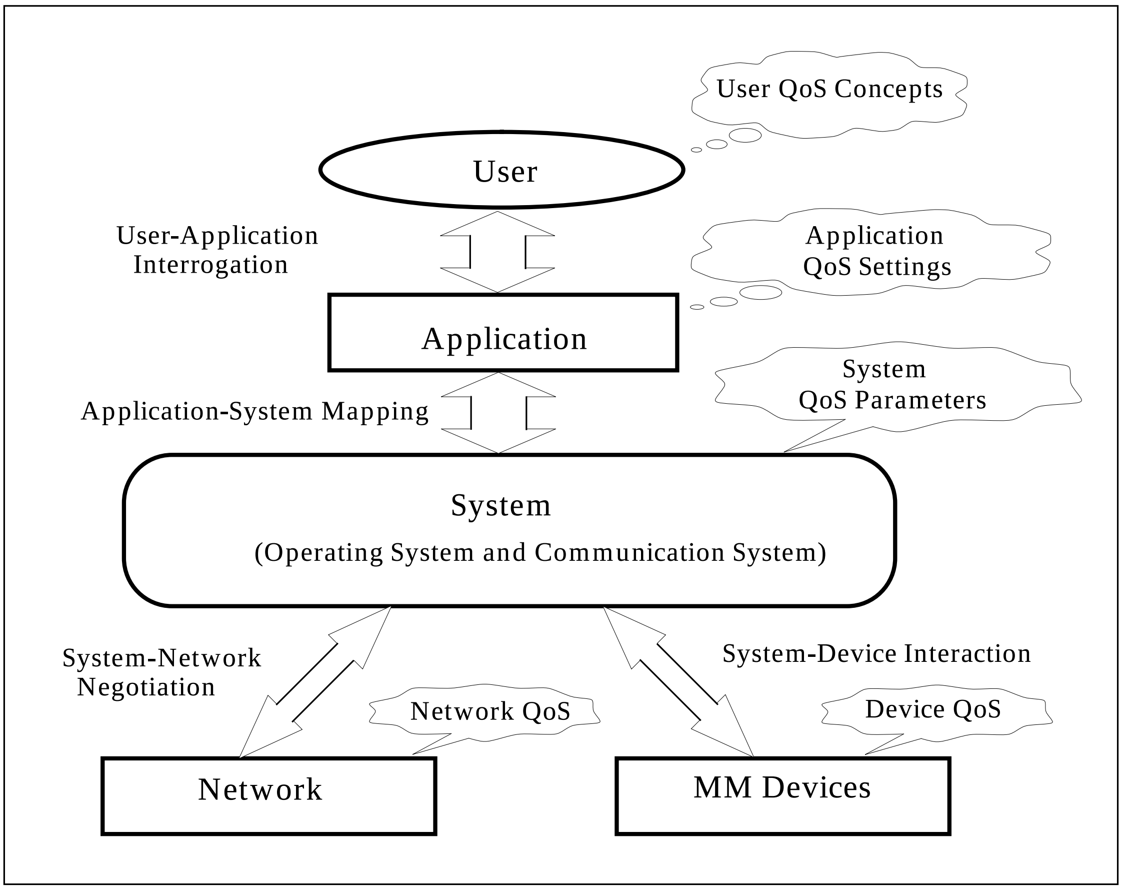
User QoS Concepts
User
User-Application Interrogation
Application QoS Settings
Application
Application-System Mapping
System QoS Parameters
System
(Operating System and Communication System)
System-Network Negotiation
System-Device Interaction
Network QoS
Device QoS
Network
MM Devices

***Figure 2-1*** **Layering, mapping and negotiation of QoS**

There are several problem domains relatd to the processing of QoS parameters:

1) Assessment and specification of the QoS requirements in terms of users' wishes or satisfaction with the quality of the application, be it performance-oriented, format-oriented, synchronization-oriented, cost-oriented or subjective-oriented.

2) Mapping of the specifications onto the QoS parameters of various system components or layers.

3) Negotiation between system components or layers to ensure that all system components agree and are able to meet the required parameters consistently.

4) Realization and maintenance of the promised QoS parameters by the participating system components or layers. This will generally involves resource management issues such as reservation and scheduling.

5) QoS requirement changes may occur due to the changes on the side of service providers or on the side of service users. In the case of a change in QoS, a renegotiation process between the participating components is usually necessary to achieve a new QoS agreement. The realization and maintenance of the new QoS agreement should then be done by all sides correspondingly.

As a matter of fact, the support of a continuous media stream should involve the whole of the stack. As partly depicted in Figure 2-1, the representation of an audio-visual stream varies at each layer. For example, the user level may see a HDTV video channel which maps down to an application-specific flow, which in turn is implemented as a composite of real-time processes by the operating system and so on. In addition to this physical variation of stream representation, each layer associates a different "view" of QoS for the stream. Each layer does its best to sustain the QoS agreed to the next layer up. If any layer is unable to maintain an agreed QoS, then a QoS degradation occurrs and should be handled.

A service provider, be it a component or a whole layer, should schematically contain the following functionality-parts, as is depicted in Figure 2-2: (a) service interface — to be used by service users; (b) admission control and negotiation — this part is responsible for schedulability test and QoS calculation and it is also responsible for the negotiation of services with other service providers; (c) resource management and execution — it is responsible for resource management and the realization of the services promised to users.

The four most important transmission and processing QoS parameters for the distributed multimedia applications are throughput, delay (local or global), jitter and reliability. As long as timing aspect on an end-system is concerned, predictable processing delay and jitter are most important.

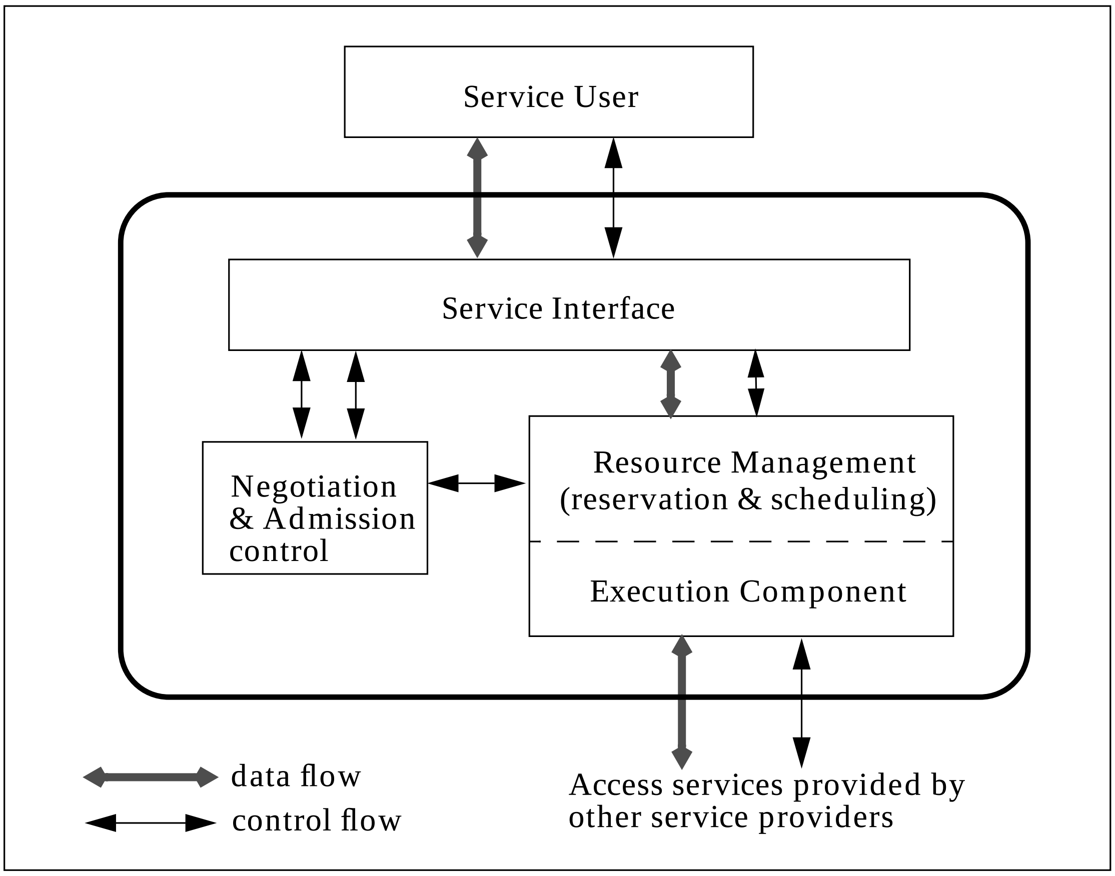
Service User
Service Interface
Negotiation & Admission control
Resource Management (reservation & scheduling)
Execution Component
data flow
control flow
Access services provided by other service providers

***Figure 2-2*** **Basic composition of a service provider**

Until recently, much work has been done with regard to QoS specification, mapping, negotiation and maintenance for communications [Hutchison94, Nahrstedt95, Campell96]. Relatively less work has been done in relation to OS functions. As further detailed in the following subsections, our work will put our special emphases on the above problem domains 4 and 5 of OS issues.

## 2.2.3 Resource management, reservation and guarantee

Services for distributed multimedia applications need resources to perform their functions. We argue that some forms of resource reservations are necessary in order to guarantee a certain level of predictability in a distributed multimedia environment.

### 2.2.3.1 “Null-reservation” not feasible for composite scenarios

Currently available workstations and high-end PCs are able to support some kinds of MM applications. They usually achieve this by allowing the MM-related applications more or less take over the machine. Their scheme of “null reservation” actually equals to “exclusive use” only for MM-related activities. As such, the multimedia applications on these systems are inflexible. They usually can not be combined with other applications. The system usually shows two symptoms. Either, other applications have direct and noticeable negative influences on the quality of MM applications. Or, the MM applications have dominated the machine so much that the other applications have almost no chances to go forward. This can be seen more clearly in the measurements of the following experiments.

***Table 2-1*** **Mutual influences of CM- and non-CM-activities**

| **Measurements** | **SmpSink Throughput** (KB/sec) | **Dhrystone Throughput** (Dhrystones/sec) |
|---|---|---|
| Scenario 1 | 115.50 | — |
| Scenario 2 | — | 71428.6 |
| Scenario 3 | 87.02 | 26881.7 |

In our simple experimental setting, a pair of simple video applications are run as source and sink on a Sun SPARCstation-4 with SunOS 4.1.3. The source application, called *SmpSend*, does the following: it captures video frames through a video camera, transforms and compresses the video frames, packetizes the frames and sends the video packets through the loopback interface to the sink application. The sink application, called *SmpSink*, performs the following: it receives the video packets, forms frames, decompresses and decodes the frames and shows the frames as a small video window. The popular benchmark program *Dhrystone* is taken as a traditional non-CM application.

In test scenario 1, only the video applications are run. The quality of video is satisfactory. The throughput of SmpSink is about 115.5 KB/sec, as can be seen in Table 2-1. In the second test scenario, The Dhrystone benchmark is run alone and a performance index of 71428.6 dhrystones/sec is achieved. In test scenarios 3, both video applications and the benchmark application are run. The result is that the video quality is quite bad, since no enough throughput has been achieved for video transport. And at the same time, it is not possible to request the processing of Dhrystone to be maintained at a level, say, 40000 dhrystones/sec. Similar observations have been made in a Solaris environment with some primitive traditional real-time support [Nieh93].

The above two symptoms are harmful to our aim MMC application systems, since we want to maintain some level of qualities for multimedia related activities and to let other activities go forward at the same time.

#### 2.2.3.2 Reasons for service guarantee and resource management

On the one hand, the capacity of a typical, currently available workstation is powerful enough to support live digital audio/video. On the other hand, this capacity is not yet enough to support multiple activities of multimedia and non-multimedia at the same time without a careful arrangement. And for our target MMC application scenario, the combinations of MM and non-MM activities are inevitable.

We are aware that some contradictory opinions on the necessity of service guarantee exist, especially in the networking fields [Deering95, Ferrari95]. In general, we hold the view that some forms of guarantee are required to support many MM applications satisfactorily. And, in order to provide service guarantees, resource reservation and management schemes have to be used.

As is usually said: “Technical advances can never catch up with the application requirements. Newer applications with higher requirements keep coming.” At any time, we can not have enough resources to meet the requirements of ever new applications. Although abundant resources can provide good services for applications in some scenarios, services will become worse in the case of higher single requirements and / or higher composite loads.

In the distant future, GigaFLOP CPUs and Terabit networks may provide acceptable performance for most application cases regardless of scheduling policies. However, in the foreseeable future, hardware resources will suffice for the continuous media applications, but only if they are scheduled carefully. In the DASH project [Anderson90b, Herrtwich92], this condition is called the “window of scarcity” as depicted in Figure 2-3. In the dark-shaded region labeled “sufficient but scare resources”, hardware resources are sufficient to handle the performance requirements of applications, but only if they are carefully allocated and scheduled in accordance with these requirements.

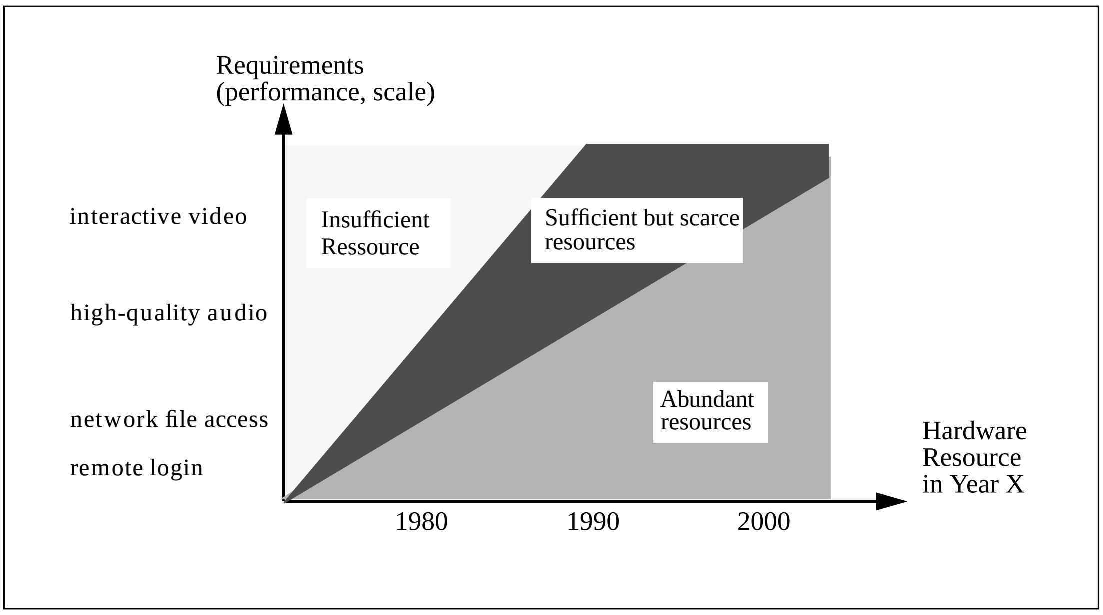

***Figure 2-3*** **Window of resource scarcity**

Resource management will always be necessary in order to main a certain level of service. When everybody gets the same best-effort service, then best-effort isn’t good enough for everybody. Practically, there are virtually nothing can be done against sustained overloads except admission control (precautionary measurement in advance) or violent expulsion (post-priori actions in case of emergency). Even on a single-user system with abundant resources, resource management is still useful to aid the user in realizing his preferences.

In a multimedia collaboration (MMC) scenario, uncontrolled resource allocation is not feasible. As shown in Figure 2-4, uncontrolled resource allocation will inevitably lead to the situations where the MM-activities are short of resources to accomplish their work or the situations where other activities can not get their proper shares on the resource or even face “starvation”.

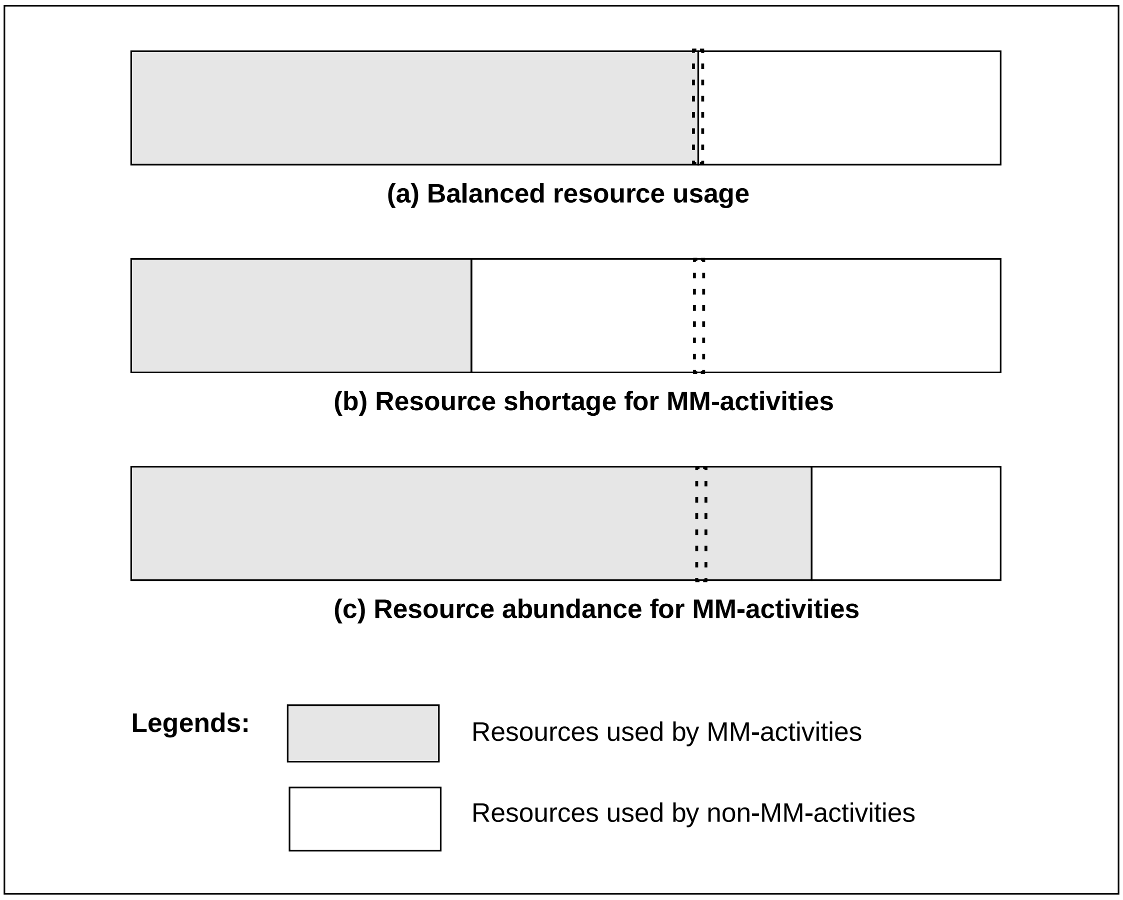
(a) Balanced resource usage
(b) Resource shortage for MM-activities
(c) Resource abundance for MM-activities
Legends:
Resources used by MM-activities
Resources used by non-MM-activities

***Figure 2-4*** **Uncontrolled resource allocation**

There are several possibilities to realize a controlled resource allocation. Rigid resource allocations for MM-activities are hard to implement and sometimes lead to resource waste. Most continuous media applications can adapt in some ranges. And we hold the view that an adaptive resource management with soft guarantee is generally the best choice (see Figure 2-5, case (c)).

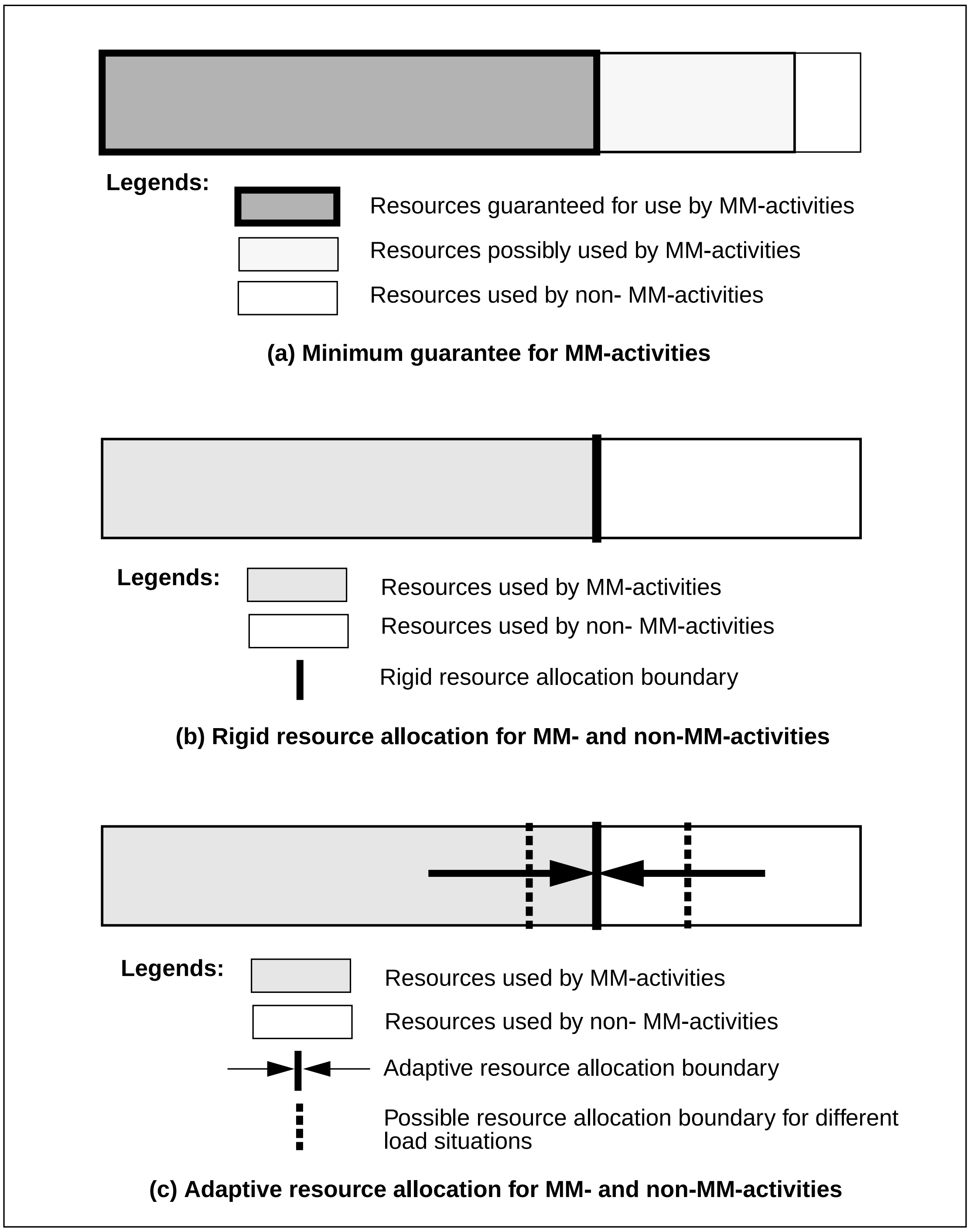
Legends:
Resources guaranteed for use by MM-activities
Resources possibly used by MM-activities
Resources used by non- MM-activities
(a) Minimum guarantee for MM-activities
Legends:
Resources used by MM-activities
Resources used by non- MM-activities
Rigid resource allocation boundary
(b) Rigid resource allocation for MM- and non-MM-activities
Legends:
Resources used by MM-activities
Resources used by non- MM-activities
Adaptive resource allocation boundary
Possible resource allocation boundary for different load situations
(c) Adaptive resource allocation for MM- and non-MM-activities

***Figure 2-5*** **Controlled resource allocation**

A multimedia application relies explicitly or implicitly on the services provided by all layers of the system. It is our belief that all layers and their main components must be able to provide some degrees of performance guarantee in order for any guarantee to be available to the upper layer users. If a layer is incapable of guaranteeing some performance bound, the layers above it cannot guarantee that bound either.

It is also necessary to offer different quality of services so that the applications can choose appropriate QoS to meet their needs. As long as guarantee is concerned, different kinds and different degrees of guarantees should be identified and provided. As is said, continuous media applications generally only need soft guarantee. For different cases, there can be different degrees of soft guarantee. Like the work on the Integrated Service Internet, we advocate the separation of concrete scheduling mechanisms from service models. A service model can possible be realized by different mechanisms.

#### 2.2.3.3 Basic ways of reservation to achieve guarantee

The traditional way in achieving guarantee is to reserve. To reserve means to separate and to control. The resources required for a guarantee should be used separately or at least in a controlled manner. The following possibilities of resource reservation can be used in different multimedia environments:

1) Null reservation — As argued above, null reservation is generally not feasible for a composite MMC scenario. But in some simple scenarios, null reservation can achieve the effect of reservation by “over-dimension”. As an example of bandwidth reservation, although a physical link might not support bandwidth reservation, bandwidth reservation might still be made in such a form that only a logical connection can be set up over a physical link so that the connection has virtually reserved the whole bandwidth of the link and this link always produces very low delays with minimal jitters. As an example of processing power reservation, extra processors can be provided for different MM devices and an application can be given enough processing power by letting it running alone.

2) Rudimentary practices — According to experiences, very simple admission control methods can be used. An example is the admission control in the

extended Ultrix OS of the Tenet group [Fisher92] where only up to 4 user real-time processes are allowed in order to avoid overload.

3) Satisfiability analysis and resource constraint enforcement — A complete satisfiability analysis of the system should be conducted beforehand or at system condition changes. During runtime, a resource enforcement should be conducted by the system in order to prevent some applications from using more resources than they claimed and which are needed by other applications. Such satisfiability (schedulability) tests should usually be done in relation to processing capacity, buffer usage as well as bandwidth needs.

An example of checking schedulability for a set of real-time CM tasks can be found in the system based on YARTOS [Jeffay91, Jeffay92]. Our scheduling framework also belongs to the last category.

### 2.2.4 Basic approaches of our work

In order to provide proper support for multimedia applications, we advocate the approach of supporting continuous media by exploiting their special characteristics. We also argue that an application-driven and application-oriented approach is the most natural approach to achieve the goal.

#### 2.2.4.1 Exploiting the characteristics of continuous media

Our central idea and starting point for supporting multimedia (especially continuous media) communications and applications is to exploit the special characteristics of them. That is, the support of OS to continuous media should exploit the special features of continuous media which are not present in or are not typical of other kinds of computer and communication applications. Some of these features can be identified as soft real-time, soft guarantee, periodicity, error-tolerance, adaptability, etc.

The following shows some of the sample working schemes and the related system effects in this regard.

For example, higher performance can be achieved by exploiting the special characteristics of continuous media communications, as is shown by the following two cases:

- **Gaining performance by using periodicity**

One common feature of continuous media communication is periodicity. That is, the continuous media shows its periodic continuity both in the sense of time and space. The periodicity of the continuous media communication can be exploited to some extent. At the application user level, periodic sending and receiving of data will result in a periodic scheduling of the corresponding tasks and threads. It is therefore possible to use the scheduling strategies which deal with periodic tasks to provide a certain degree of guarantee for their processing. At the lower level, interrupts resulted from CM communication can be partly spared if this periodicity can be used. By exploiting the periodicity at interrupt level, the processing overhead can be reduced and be virtually controlled. This is of special importance since the low-level interrupts have usually taken a large part of the processing capacity of an ordinary workstation.

- **Gaining performance by turning off checksum computation**

Since continuous media communications can tolerate a certain degree of transmission errors in many cases, it would be possible to spare the cost of checksum calculation over the user data.

It should be noted that turning off checksum protection in a wide area context seems unwise without considerable study, since a number of gateways are involved and an end-to-end corruption becomes much more possible. But in many cases it would surely be advantageous to turn off the checksum computation safely for multimedia communication in order to get a clear performance improvement. The following shows some concrete examples of performance gains by turning off checksum computation. The performance enhancements are usually significant.

In the environment reported in [Kay93], the checksum computation time of UDP messages is responsible for almost 50% of total processing time for large messages (larger than 8192 bytes). The computation of the checksum for user data has been designed to be optional in some contemporary protocols (for example, the NOERR mode of the XTP protocol). In our experiences with XTP and XTP-lite, for example, the performances of data transmission using XTP have been clearly improved at the user-application level if the user data checksum in XTP packet has been turned off [Fan93, Fan95c].

- **Soft real-time, soft guarantee and adaptive service support**

Our work on the realization of a predictable soft real-time framework and the realization of an adaptive service supporting model are all based on the idea of exploiting the special characteristics of continuous media. Further texts of the dissertation will present the details.

#### 2.2.4.2 Application-driven and application-oriented approach

In our opinion, it is a natural and perhaps the best method to do resource reservation and other OS support in an application-driven approach. The QoS requirements of an applications dictate the resources required by the underlying system components which service this application.

- **Mapping the QoS requirements of upper layer down to parameters supported by lower layers**

Without the basic support of lower layers, the guarantee at higher layers are very difficult, if not impossible. Some QoS parameters at the upper layer applications will eventually have to be mapped on to the QoS provisions of the supporting operating systems and networks. For communications, the QoS support of underlying networks should be used in the end. Networks are evolving towards the provision of basic performance guarantee. By using ATM signalling protocol, for example, resources will be reserved in the ATM network such that the connection will be guaranteed with its bandwidth, delay etc. For processing capacity needs, the timing requirements of the applications will have to be embodied in the form of timing parameters of the real-time processes which implement them.

- **Reserving protocol processing power by application-driven protocol processing**

As can be seen in the playback scenario abstraction, the MM data packets need not be processed as soon as possible, they need only to be processed in time — before their playback points. In Chapter 8 “Predictable protocol processing”, we will see how a delayed, object-activated processing model can make it possible to reserve protocol processing power by application-driven protocol processing.

### 2.2.5 Main emphases of our work

Our work will only deal with endsystem OS support for multimedia.

It is clear that the proper resource managements for all resources in the endsystems as well as the routers are needed in order to provide reliable QoS to users of a multimedia communication system. A key element of the current challenge is the provision of performance-oriented QoS parameters such as throughput, delay and delay jitter on top of asynchronous networking systems. The provision of multimedia QoS parameters has to cope with the non-deterministic nature of mainly two components:

1) the communication environment — asynchronous networking systems which cause probability distributions of performance related parameters of the information flow; and
2) the OS or other form of run-time support — real-time and non-real-time operating system scheduling facilities which can only partly support the bounded processing delay for communication requirements.

To meet the requirement of distributed multimedia QoS support, reservation schemes should be able to set up and reserve network resource such as bandwidth and end-to-end delay of different paths over the whole way of a connection. Otherwise, numerous dropped and delayed packets are unavoidable. In this work, we do not tackle the problems related to the communication environment. For our purpose, we assume a networking environment with similar functionality to an Integrated Service Network [Clark92, Shenker95a, Shenker95b, Braden96] or a TENET network [Ferrari92]. That is, the OS should be allowed to request QoS parameters on network connections and the network should try to maintain these QoS parameters or at least give a notification to the OS to indicate QoS violations if the maintenance of the QoS parameters is no longer possible. However, our schemes will *not* assume that the networks can provide any rigid and absolute real-time support.

As stated in *Section 2.1 A brief state-of-the-art survey*, OS MM supports encompass all components of the QoS architectures, all system resources and all areas of OS. Needless to say, it is impossible to handle all these issues in our work. Our work has been centered around the real-time aspect of the multimedia system. We mainly deal with the most critical resource of the system — processing possibility of MM activities, i.e., the scheduling of multimedia processing. The other resources should be correspondingly managed in an pplication- and timing-driven manner. Our emphases are not on the definitions or mapping of related QoS requirements, but the actual realization of these QoS requirements. Such work includes resource (especially CPU) reservation, QoS maintenance and violation handling.

Our work has been organized under the goal of realizing a predictable soft real-time framework. Prototype implementation and simulation evaluation have been conducted in the scope of an internal project effort called “MMOSS”, which stands for “multimedia operating system support” [Fan94a, Fan95a, Fan96b]. Part of the work has also been reflected in some milestone reports for a TUB-OKS-Siemens Cooperation Project [Fan94b, Schatzmayr94, Fan95b].

## 2.3 Summary of the chapter

Through a brief survey, the state-of-the-art of OS support for multimedia is presented. The need to accommodate continuous media delivery and processing has already affected every part of the operating system. We take a glance on the work in several areas of current active research and development: OS kernel support for multimedia, file system and file storage server for multimedia as well as specific network access and communication protocols. We have also discussed the implementation schemes of typical applications so that we can better understand the applications’ behavior and needs and can thus provide the adequate support for them.

We have also justified our system models and emphasized our main working points. Our work on OS support aims at fully integrated multimedia end-systems because of their many technical and future-oriented advantages. Our QoS architecture deals with all layers and components of the system. The necessity for service guarantee and resource reservation is also justified and some basic ways of reservation are presented. The basic idea of our work is to support continuous media by exploiting the special characteristics of them. To achieve this goal, we take an application-driven and application-oriented approach. Our emphases are not on the definitions or mapping of related QoS requirements, but the actual realization of these QoS requirements. Our work has been centered around the real-time aspect of the multimedia system and has been organized under the goal of realizing a predictable soft real-time framework. We use theoretical analyses, simulation evaluations or experimental implementations to validate our models and algorithms.

# PART II

# Using a Soft Real-Time Framework

# Chapter 3

# A soft real-time framework as a solution

In Part 1, we have briefly argued that real-timeliness is a prerequisite for multimedia systems. While it is often assumed that continuous media is challenging because it imposes the highest data rates or processing requirements on the system, this may not necessarily be the most critical factor. Graphics or transaction processing may well impose larger loads on a larger range of system components. It is actually the timing requirements and the guaranteed throughput required for continuous media that force specific consideration in designing operating system supports for continuous media. In addition, the high data rate or large processing capacity requirements can be more easily “scaled out” with the rapid development in the hardware fields. In contrast, the timing and guarantee requirements demand solid work on a theoretically-sound technical basis.

In an integrated multimedia system, the operating system must be able to fully support and control the multimedia-related activities. In order to make the whole system predictable as well as flexible, we argue that all the system activities should be controlled by a soft real-time framework. The soft real-time framework is responsible for controlling networking services as well as end-system services which access and process the media data on behalf of the users.

This chapter begins by arguing that the CM applications and communications need soft real-time scheduling and that the current systems do not provide the appropriate support. Further analyses are provided to show that CM applications and communications posses a set of “soft” features which can be accommodated in a soft real-time framework. Finally, the soft real-time framework is sketched with its main components identified.

## 3.1 CM applications and communications need real-time scheduling

The applications involving continuous media can appear in many forms. In addition to the multimedia collaboration (MMC) scenario, which is our main problem domain, some other examples can be found in:

- Playback of full-motion video and audio recorded in digital form on compact disks;
- Audio-visual digital teleteaching, where the transmissions are generally unidirectional multicast;
- Full-motion computer animation accompanied by synthesized sound.

Suppose these applications are to be supported on a new desktop multimedia computer system, then it should be clearly a multi-tasking window system, which is to support as many concurrent instances of the above-mentioned tasks as is feasible. Continuous media applications demand the processing of audio and video data in such a manner that humans can perceive these media in a natural, non-artificial way.

These CM applications have their real-time constraints which should be satisfied. For example, it should be ensured that sounds and videos do not have glitches caused by late arrival of packets or shortage of processing, etc. That is, the quality of service is impaired if CM data isn’t delivered and processed on time. Therefore, multimedia systems are real-time systems, since they are systems “in which the correctness of computation depends not only upon the results of computations but also upon the time at which the outputs are generated” [Stankovic88, Zalewski93].

By the way, one can even find the cases where CM applications have genuine hard real-time requirements. One such example is an embedded multimedia system such as a digital video recorder, in which hard real-time techniques should be used otherwise video frames could be dropped and a consumer would, of course, require that it be fixed. Such cases are *not* our main concerns here.

Most of the continuous media applications which interest us do not have hard and stringent deadline requirements. By video conferencing, for example, small disturbances in video can be tolerated. The most continuous media applications are therefore soft real time applications, where "system service and performance are degraded but not destroyed for failure to meet some response time constraints." They need not be treated as hard real-time systems because they need not "be designed to meet given deadlines under all circumstances to avoid high cost or even danger" [Zalewski93, Audsley91, Mercer92].

Anyway, developers of multimedia software should incorporate real-time design and analysis techniques in order to build systems that function predictably and reliably.

### 3.1.1 Why existing systems are not fully feasible

Clearly, the traditional time-sharing operating systems can not provide adequate support for real-time multimedia applications and communications. In those systems, some form of "fair-share" should be granted to all system activities so that the more processes there are, the more seldom each process gets executed on the average. This, of course, can not meet the needs of multimedia applications which have real-time constraints and processing capacity requirements. For example, standard Unix scheduling favors I/O intensive processes by adjusting their scheduling priorities correspondingly, while software-based codecs may not get sufficient computational resources.

In most commercially available real-time operating systems, emphasis was placed on fast interrupt latency, fast context switching, and small preemptable kernel. A few commercial systems also provide synchronization primitives which avoid the priority inversion problem [Fiddler89, Hildebrand92, Mukherjee93]. They have one in common: they usually provide a careful scheduling policy for delay bounds, not for throughput guarantees; and the real-time feature has to be used by each particular application in an ad hoc manner. In addition, some main features of these real-time OS such as short interrupt response latency and fault-tolerance are not very useful for MM systems. For example, for multimedia communication itself, interrupt handling latency is not a main problem, at least not a central critical problem. The reason is that a reasonable amount of buffer is always necessary to compensate the jitter. The same can be said with fault-tolerance. In multimedia systems for

MMC, fault-tolerance need not be emphasized. On the other hand, these existing systems lack some important features needed by MM applications. For example, there are usually no direct supports for explicitly expressing timing constraints such as processing throughputs, periodicities, etc. Under transient overload, users may loose control over which tasks should be favored to complete their computations and which tasks should be delayed or even aborted. By using these real-time operating systems, it is difficult to construct flexible multimedia systems.

### 3.1.2 A whole system concept is needed

To support multimedia applications such as MMC best, a multimedia supporting system should support soft real-time development methodologies and the whole end-system should be controlled by a soft real-time framework.

It should be possible for multimedia developers to analyze the real-time behavior of their programs in order to characterize the circumstances under which the programs would function correctly. The real-time methodologies should also allow programmers to carefully isolate the real-time behaviors of each component in the system in order to have a good control of different application mixes and to facilitate debugging.

It should be noted that all the activities related to continuous media applications should be supported by the real-time mechanisms. The real-time support of kernel activities alone is not enough. Under heavy or even normal load, application processes which consumes the CM information should also be run in real-time manner, otherwise the real-time constraints of the whole activities can not be satisfied. For example, even if the data can be delivered at the transport service access points by real time, it will still be useless, if the application processes are delayed and can not make use of the delivered data in time. We hold actually the point of view that a kind of end-application-driven scheduling strategy should be promoted.

In a normal MMC application scenario, the workload contains a mixture of real-time, interactive, and batch applications. Multimedia OS support should accommodate MM as well as non-MM activities.

## 3.2 “Soft” features of CM applications and communications

Since the continuous media applications and communications posses many real-time (RT) features, the CM supporting systems need not and should not do their design from scratch. Rather, the design should be based on the currently available real-time techniques. The main concern here is just how to select, modify, extend and combine these techniques to make them meet the needs of the specific soft real-time features of the continuous media. In the following, we try to identify and formalize some of these soft features.

### 3.2.1 Modeling CM activity as play-back application

The continuous media data has its origin at sources like microphones, cameras and files. From these sources the data is transferred to destinations (or sinks) like loudspeakers, video windows and files located at the same computer or at a remote station. On the way from a source to a sink the digital data is processed by at least some type of move, copy or transmit operation. Therefore in this data manipulation process there are always many resources which are under control of the operating system. The resource management and the respective scheduling must be performed according to the real-time demands of the continuous media applications. By examinating the whole way from the source to the sink, we can again divide the way into several stages. The sink of an intermediate stage is again the source for the following stage. It is important to note that the goal of the whole process is to “play-back” at the sink the CM data generated by the source in a certain acceptable manner. This leads to some requirements concerning delay, jitter etc. In order to put the delay and jitter under control, each stage on the way should usually assert some forms of control on the delay and delay jitter. Of course, the end-to-end delay and delay jitter matter most.

The whole way from a source to a sink or some stages of it can be abstracted as some forms of “play-back” scenarios. It has been natural to model CM communications as play-back applications [Clark92, Kurose84, Partridge91, Mathur93]. This applies readily to other stages on the way to transfer CM data from a source to a sink. CM data are usually generated at the source periodically and continuously and are then consumed at a destination periodically and continuously, regardless whether the source and the destination reside locally with each other or are connected by a communication network. Here, one or both of the source and destination can be CM devices or some forms of intermediate stops.

Let us first formalize the notion of play-back point to some extent. (In the literature, “play-back” is sometimes called “play-out”.) For sake of simplicity, we use the term “packet” to denote either real CM data packet or virtual processing event related to CM data.

Let the end-to-end delay of a packet **i** be $D^i$, which is the delay experienced by the packet from the time it is created at the source side to the time it is consumed at the destination side. The packet delay consists of approximately two components, a fixed and a variable component. The fixed component of the total delay, denoted $D_{fixed}$, is principally identical for all packets and includes, for example, propagation delay and other fixed processing overheads. The variable component of the delay, denoted $D^i_{variable}$, is the queueing delay experienced by the packet at various stage of its processing or transmission path. We have, therefore,

$$D^i = D_{fixed} + D^i_{variable} \text{ and } D^i \geq D_{fixed}.$$

The playback delay of packets at the destination is defined as

$$D_{playback} = D_{fixed} + D_{slack}.$$

Let us denote the ideal generation time of packet **i** at source as $t_s^i$, the actual packet generation time as $t_g^i$ (possibly $t_g^i \neq t_s^i$, especially for the intermediate stages). Then $D_{slack}$ should be so chosen that for a high percentage of packet **i**: $t_r^i \leq t_d^i$, where $t_r^i$ ($t_r^i = t_g^i + t_{delay}^i$) is the actual ready time of packet **i** at destination and $t_d^i$ ($t_d^i = t_s^i + D_{playback} = t_s^i + D_{fixed} + D_{slack}$) is the ideal destination consumption time (playback time or playback point) of packet **i**. Only if $t_r^i \leq t_d^i$ can the packet **i** be used by destination in time. If a packet does not arrive by its playback time, i.e. $t_r^i > t_d^i$, then it can not be used for its playback in time and there will be a break in the continuity of the playback. For example, packet 4 in Figure 3-1 is useless because its arrival time is after its playback point.

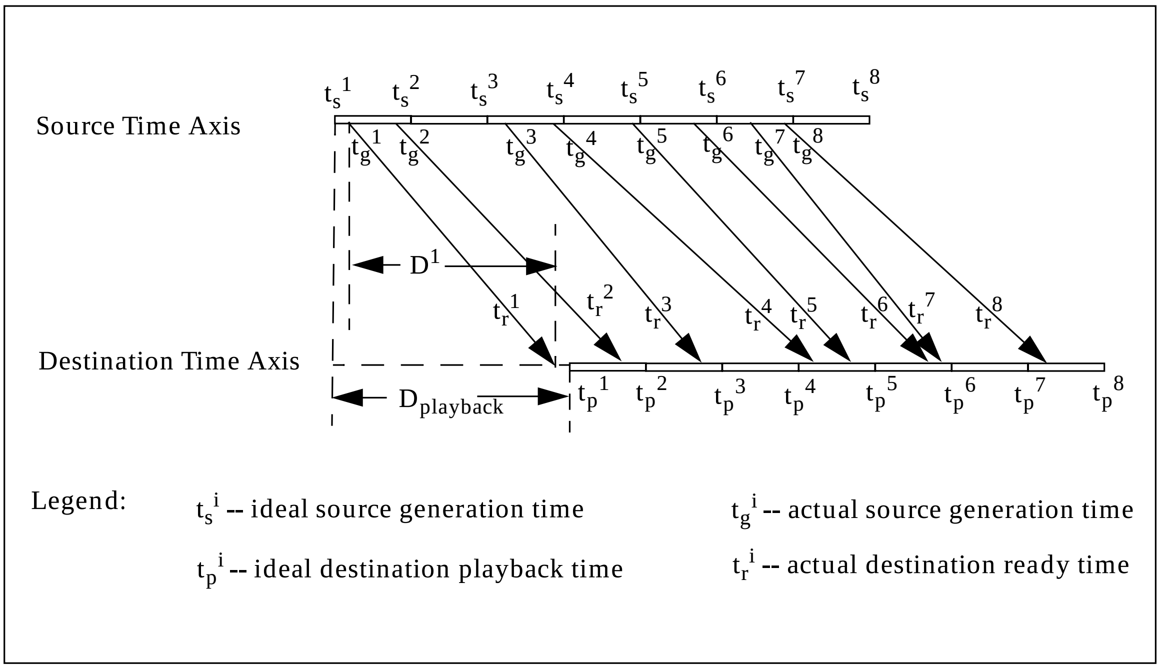
$t_s^1$ $t_s^2$ $t_s^3$ $t_s^4$ $t_s^5$ $t_s^6$ $t_s^7$ $t_s^8$
Source Time Axis
$t_g^1$ $t_g^2$ $t_g^3$ $t_g^4$ $t_g^5$ $t_g^6$ $t_g^7$ $t_g^8$
$D^1$
$t_r^1$ $t_r^2$ $t_r^3$ $t_r^4$ $t_r^5$ $t_r^6$ $t_r^7$ $t_r^8$
Destination Time Axis
$D_{playback}$
$t_p^1$ $t_p^2$ $t_p^3$ $t_p^4$ $t_p^5$ $t_p^6$ $t_p^7$ $t_p^8$
Legend:
$t_s^i$ -- ideal source generation time
$t_g^i$ -- actual source generation time
$t_p^i$ -- ideal destination playback time
$t_r^i$ -- actual destination ready time

***Figure 3-1*** **Time diagram for a section of playback scenario**

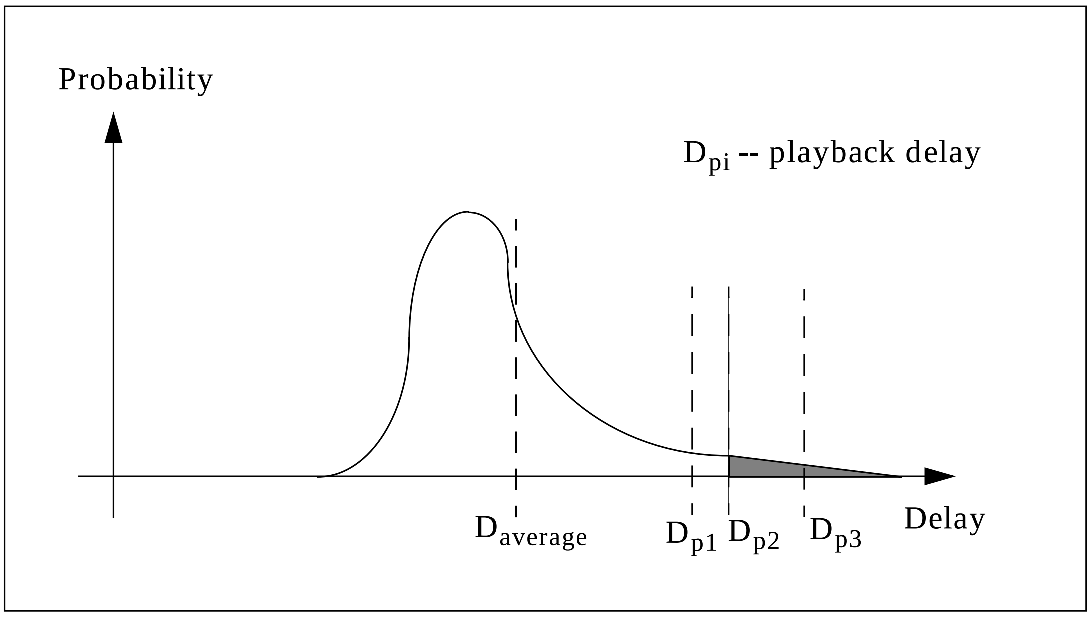

***Figure 3-2*** **Delay distribution and the choice of playback delay**

In the delay distribution shown in Figure 3-2, the packets with a delay larger than $D_{p2}$ is represented in the shadowed area. If the playback delay is chosen to be $D_{p2}$, then the packets represented by the shadowed area can not be used for playback and are therefore “meaningless” or “lost” for the playback. If the playback delay is moved forwards to $D_{p3}$ or backwards to $D_{p1}$, then the shadowed area becomes smaller or larger correspondingly. This means less or more “meaningless/lost” data correspondingly. This illustrates the effects of different choices of playback delays, where $t_{p1}$ apparently causes more packets to miss their playback points than $t_{p2}$.

Suppose the probability density function (pdf) of the end-to-end delay to be f(x) and suppose the playback delay is set to D. For a simplified representation, the percentage of the lost packets and the percentage of the packets which can be successfully played back are then respectively:

$$P_{loss} = \frac{\int_{D}^{\infty} f(x)dx}{\int_{0}^{\infty} f(x)dx} \qquad \text{(E-3-1)}$$

$$P_{success} = 1 - \frac{\int_{D}^{\infty} f(x)dx}{\int_{0}^{\infty} f(x)dx} = \frac{\int_{0}^{D} f(x)dx}{\int_{0}^{\infty} f(x)dx} \qquad \text{(E-3-2)}$$

If the delay is of a constrained range of the form [min, max], for max ≥ D ≥ min, the percentage of the lost packets and the successful packets are then respectively:

$$P_{loss} = \frac{\int_{D}^{max} f(x)dx}{\int_{min}^{max} f(x)dx} \qquad \text{(E-3-3)}$$

$$P_{success} = 1 - \frac{\int_{D}^{max} f(x)dx}{\int_{min}^{max} f(x)dx} = \frac{\int_{min}^{D} f(x)dx}{\int_{min}^{max} f(x)dx} \qquad \text{(E-3-4)}$$

The relationship between the success rate and the play-back delay is a key index in a CM application. The whole goal is to make a trade-off to achieve the best total system effect. On the one hand, we have to keep the playback delay $D_{playback}$ small in order to reduce the negative effects induced by long end-to-end latency. This can be the reason that we might choose $D_{p2}$ instead of $D_{p3}$ in the scenario shown in Figure 3-3, although $D_{p3}$ is long enough to ensure that all packets arrive before their playback points thus providing a better playback quality. On the other hand, we have to keep the playback delay $D_{playback}$ large enough in order to maintain a continuous playback by ensuring that most of the packets arrive by their playback time. This can be the reason that we might choose $D_{p2}$ instead of $D_{p1}$ in Figure 3-3, since $D_{p1}$ provides a shorter end-to-end delay at the cost of too many packet loses.

It should be noted, however, that the playback delay need not be kept constant all the time. It is actually a normal practice for the continuous media applications to adjust their playback delays according to the changing situations. More on this will be detailed in *Section 3.2.2 Flexible features*.

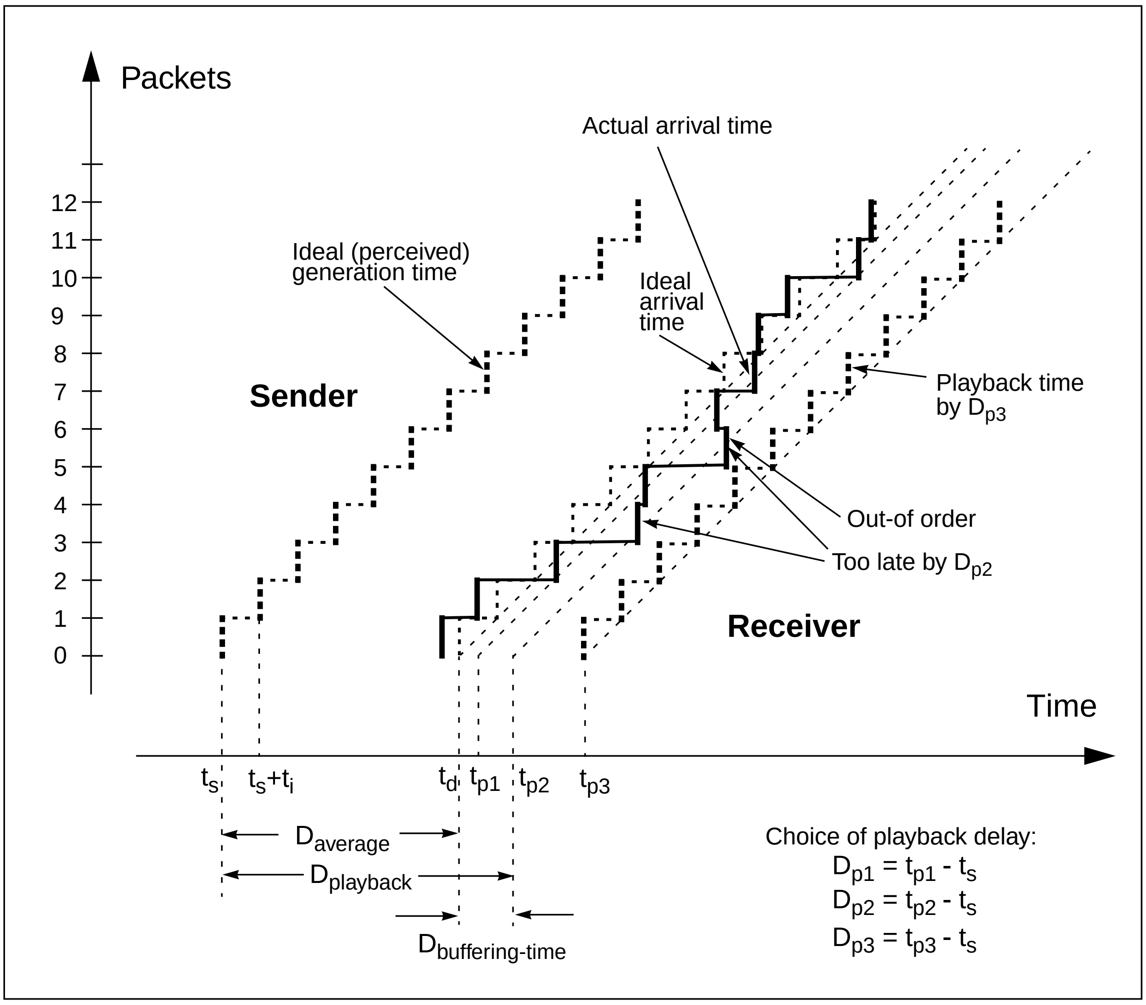

***Figure 3-3*** **Generation and reconstruction of a stream as seen by the receiver**

In an audio conferencing, for example, the end-to-end playback delay should usually be kept under 300ms in order to make a natural real-time conversation possible (CCITT G.114 recommends a 400ms upper bound for most applications). At the same time, the end-to-end playback delay can not be so small that many of the data packets can not arrive in time thus making the voice incomprehensible for the listener. The loss tolerance of real-time audio transmission can vary in the range of 1% to 10% under different conditions [Schulzrinne96]. The human beings are usually even more tolerant to a video distortion than to an audio distortion [Steinmetz95].

- **A side-note on isochron and synchronization**

As implied above, one of the properties of continuous media applications is the property of isochron. That is, the playback of voice and video data streams should be isochronous in that their users may be sensitive to the variation in inter-arrival times between data delivered to the ***final*** output device.

Intuitively, jitter is the measurement of the non-steady fluctuations / variations of the packet arrival time. For sake of simplicity, a delay jitter is defined as the variation of packet delays relative to a reference packet delay, as is shown in Figure 3-4. Alternatively, one can also use inter-arrival jitter as a measurement. In RTP, for example, the inter-arrival jitter is defined to be the mean deviation (smoothed absolute value) of the difference in packet spacing at the receiver compared to the sender for a pair of packets.

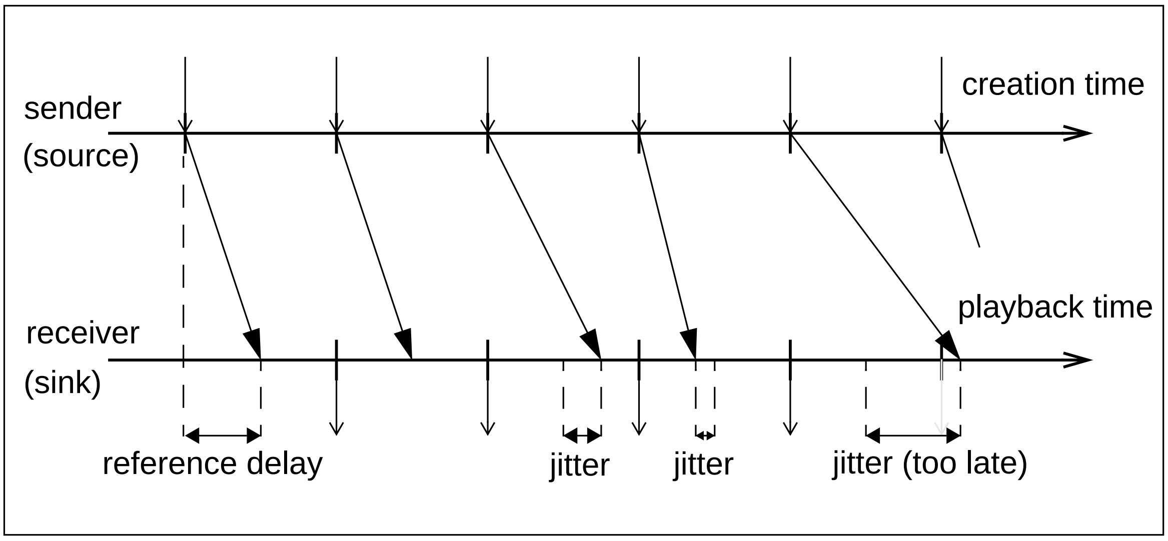

***Figure 3-4*** **Delay jitter as a result of delay variations**

Note that the factor of delay jitter is considered implicitly in the above modeling scheme. The playback delay is used to compensate the delay jitter to a large extent. The playback delay $D_{playback}$ should usually be larger than the average delay $D_{average}$ so that the most packets can be processed in time to be played back. Intuitively, the greater the jitters are, the larger the $D_{playback}$ should be than $D_{average}$. In order to set a reasonable value for $D_{playback}$, it is helpful to know the average jitter.

The packets arrived earlier than their due playback time usually have to be buffered up to their playback time. We will turn back again to the issues of buffering the earlier-arriving packets and the support of isochronous playback later in Chapter 5.

Actually, the temporal aspects of CM data contribute to their semantics and they result in a need for synchronization of threads that present CM data to users or user processes. The following two forms of synchronization can be identified:

- Intra-stream synchronization, i.e., within a stream of CM data. This is just the property of isochron explained above. The consecutive values of a CM stream must be presented at regular intervals.
- Inter-stream synchronization. This may be needed between different streams of CM data as well as between CM and discrete media (DM) data. If several streams of CM or DM data are semantically related, their values have to be presented in a synchronized manner. An example is “lip-synchronization”, where it is necessary to synchronize the spoken voice with the movement of the speaker’s lips. Another example is to incorporate DM data in CM data as in the case of showing subtitles in a movie. The processing and output of such DM data has to occur when certain values of the CM data sequence are reached. In the multimedia collaboration environment, the availability of CM data such as audiovisual streams and the availability of DM data such as common documents should also be synchronized loosely.

We are aware that the above playback scenario has just modeled the intra-stream synchronization aspect of the CM applications. (More considerations with this regard will be given in *Section 5.4.4 Jitter of processing* and *Section 5.5.3.2 Interfacing continuous media I/O*.)

We hold the view that it is neither necessary nor possible to provide support for all temporal aspects of continuous media. The aspect of inter-stream synchronization can be realized by the applications with the help of transport, session and application protocols. The direct support for intra-stream synchronization of CM processing can control the variations in processing delay jitter thus providing a good basis for inter-stream synchronization. For discrete media processing, our process framework will provide a kind of virtual soft real-time support to make sure that the

DM-related processing can progress with the CM-related processing together.

### 3.2.2 Flexible features

In comparison to hard real-time applications, the nonrigidity or flexibility of continuous media applications and communications can be seen in several aspects. (Of course, we also keep in mind that not all the CM applications have the same flexibilities or the same degrees of flexibilities.)

First, some losses of CM data packets, some omissions in processing CM data packets or CM-related events are acceptable. Although the permissible losses or omissions vary from applications to applications, human eyes and ears apparently can smooth some glitches from missing samples or events.

Second, we can identify three kinds of possible flexibilities (adaptabilities) which can be exploited. That is, rate, data volume and playback delay of CM applications are in many cases adaptive and can be adjusted in a certain range.

Let us look at some examples. The first example is a video transmission scenario. If the effect of “slow motion” can be tolerated by viewers for some time then the number of frames transmitted in a second can be lowered for some time. The second example is the transmission and processing of video packets encoded in MPEG. By choosing not to transmit or process some P or B frames (or better, not to encode them in the first place), the data volume can be decreased. The third example is the *VAT* voice-conferencing system developed by LBL and in experimental use over the MBONE of the current Internet [Casner92, Partridge94]. Although the current Internet has no support for bandwidth reservation and real-time transmission, *VAT* can still deal with the problem of varying delays of voice packets by changing the playback point, during the conversation, in response to the network delays it observes. Voice samples are time-stamped at sending side and replayed at receiving side. If *VAT* discovers that a lot of voice samples are arriving late, it increases the playback point. If *VAT* observes that all the voice samples are arriving well before they are played, it moves the playback point back. *VAT* shifts the playback point during intervals of silence in the audio stream so that the changes are almost invisible to listeners.

Concerning the adjustment of the playback delay, one can be flexible in the case of audio playback as well as in the case of video playback. Playback delay adjustments done at the talkspurt boundaries are usually unnoticeable by the listener. And one can as well discard some video frames or to replay some video frames to speed up or to slow down time.

As a consequence, as long as scheduling is concerned, continuous media applications and communications usually posses the following features:

1) A high degree of guarantee is required but a strict hard guarantee is not needed. That is, some violations of timing constraints can be tolerated.
2) A good degree of CPU-usage should be achieved. That means, for example, the schedulability's test and the scheduling mechanisms for real-time processes or threads should not be too time-consuming; sometimes a trade-off between efficiency and optimum should be made.
3) Real-time processes as well as non-real-time processes should be accommodated in the same framework to support a composite multimedia system.
4) A certain degree of elasticity in describing the real-time properties of a real-time process at creation time should be allowed. For example, the worst execution time of the process need not be the exact real upper bound of the process execution at run time.

As is evident, one of the key points is how to exploit the possibility of sacrificing absoluteness for a higher efficiency and utilization, since multimedia systems are generally no hard real-time systems. And at the same time, a certain degree of guarantee of service provision should be maintained. Briefly speaking, resource allocation should be done in order to achieve a certain degree of predictability, but we only need to allocate less than peak resources and rely on statistical multiplexing of multiple applications and the felxibilities of these applications to get a satisfactory probabilistic guarantee.

## 3.3 Sketch of the soft real-time framework

To summarize, the above analyses show that the real-time requirements of the continuous media are different from the real-time requirements in a safe-critical hard real-time system. They are surely “softer”. (They can be seen as more favorable [Steinmetz95] or more complicated [Mullender94] than hard real-timeliness, depending on the observer’s perspective angles.) It is advantageous to have some degree of real-timeliness in the scheduling of continuous media, but the misses of deadlines are tolerable by the CM processes to some extent.

All in all, it is only necessary to achieve a good approximation of the predictability guaranteed by the hard real-time scheduling theory. I.e., a certain degree of “skew” is allowed. We call such a framework a soft real-time framework. In a general sense, the framework comprises not only the operating system as its center, but also the underlying hardware supporting components and the overlying software packages which make the support of continuous media complete.

### 3.3.1 Practical meanings of soft real-time

The soft real-time needed for the support of continuous media has several practical meanings which will be taken into consideration when we design and realize our soft real-time framework.

- **Different degrees of soft real-time**

Not all the CM applications have the same flexibilities or the same degrees of flexibilities. This leads to different degrees of soft real-time needed to support them. More specifically, this means different degrees of real-time requirements for different application domains, different applications and different environments. For the design and implementation, different degrees of soft real-time means different degrees in the strictness of soft guarantee, admission control, monitoring and enforcement strategies, etc.

- **Two aspects of soft real-time**

The real-time features of a multimedia system can be soft in mainly two aspects. On the one hand, the user processes can be (and should be) allowed to misbehave to some extent and be allowed to violate some timing agreements from their side. For example, the user processes may try to use more processor time than they claimed and reserved. On the other hand, the supporting system can also be (and should also be) allowed to misbehave to some extent and be allowed to violate some timing agreements from their side. For example, the supporting system may not conduct a very precise accounting on the user processes' resource usage and the supporting system may "steal" some time budget from a user process when the system happens to have to handle a burst of system events.

The tolerances of timing violations from both the side of the system and from the side of the user processes have also their practical meanings and consequences. First, different guarantee degrees can be supported as explained above. Second, it is possible for the system and the user processes to state their timing specifications flexibly and timing overflows/exceptions can be handled in some "soft" and "graceful" manners. The handling methods for timing exceptions or violations are necessary, since we do not require a rigid timing specification from the multimedia processes and not-so-often timing violations both from the side of user and from the side of the system are allowed.

### 3.3.2 Contents of the soft real-time framework

Generally speaking, enhancements in both the endsystem hardware and software are needed in order to meet the time-constrained high processing requirements of the multimedia applications. In this sense, the soft real-time framework should comprise not only the operating system as its center but also the underlying supporting components and the overlying software packages to make the support of continuous media complete. Very often indeed, certain hardware configurations are needed in order to make the software aspect of the soft real-time support easy.

For the part of OS, our soft real-time framework proposal consists of a process framework for categorizing processes, some timing enforcement models and some base real-time scheduling and handling schemes. The processes are categorized so that they will receive different treatments with regard to resource allocations. The timing enforcement models deal with the issues of enforcing timing contracts between systems and user processes in different environments. Base real-time scheduling schemes are used to schedule the system activities and the base real-time handling schemes are used to deal with the situations of timing overflow.

These components are designed to work together to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system is run in a more or less predictable manner. The following chapters will present some concrete design, implementation and evaluation of these components.

We are aware, of course, that a multimedia application is dependent on several kinds of resources in its whole life time and that the processing capacity is only one of the most important active resources. Real-timeliness and predictability of the system as a whole can only be achieved if all system resources and activities involved are managed in a predictable manner. Coupled with some concrete design of soft real-time scheduling and handling methods, Part III and Part IV will investigate, among others, realization issues such as application- and timing-driven resource reservation, interfacing continuous media I/O and predictable protocol processing.

## 3.4 Summary of the chapter

One of the key points of the feasible multimedia support is how to exploit the possibility of sacrificing absoluteness for a higher efficiency and utilization, since multimedia systems generally do not need hard guarantee. At the same time, a certain degree of guarantee of service provision should be provided in order to maintain a certain degree of application-perceived quality. Correspondingly, resource allocation should be done in order to achieve a certain degree of predictability. But we only need to allocate less than peak resources and rely on statistical multiplexing of multiple applications and the felxibilities of these applications to get a satisfactory probabilistic guarantee.

Continuous media communications and applications certainly require real-time support. But the real-time requirements of the continuous media are different from the real-time requirements in a safe-critical hard real-time system. They are surely “softer”. In comparison to hard real-time applications, the nonrigidity or flexibility of continuous media applications and communications can be seen in several aspects.

First of all, some losses of CM data packets, some omissions in processing CM data packets or CM-related events are acceptable. Although the permissible losses or omissions vary from applications to applications, human eyes and ears apparently can smooth some glitches from missing samples or events.

In addition, we can identify three kinds of possible flexibilities (adaptabilities) which can be exploited. That is, rate, data volume and playback delay of CM applications are in many cases adaptive and can be adjusted in a certain range.

In continuous media-related applications, the whole way from a source to a sink or some stages of it can be abstracted as some forms of “play-back” scenarios. The relationship between the play-back success rate and the play-back delay is a key index in a CM application. A trade-off should usually be made in trying to achieve a short play-back delay (and thus a short end-to-end delay) and a high play-back success rate (and thus a high play-back quality) at the same time.

The traditional time-sharing operating systems and the commercially available real-time operating systems can not provide flexible soft real-time supports for the multimedia applications and communications. To support multimedia applications such as MMC best, a multimedia supporting system should support soft real-time development methodologies and the whole end-system should be controlled by a soft real-time framework.

In realizing such a soft real-time framework, different degrees of soft real-time should be considered for different applications and environments. Soft real-time also implies the tolerance of some timing violations both from the side of the system and from the side of user processes. This feature has also to be exploited in the design and realization of a soft real-time framework.

As a system framework, a soft real-time framework will inevitably involve both hardware and software components, with the operating system as its center. For the part of OS, our soft real-time framework proposal consists of a process framework for categorizing processes, some timing enforcement models and some base real-time scheduling and handling schemes. As will be detailed in the following chapters, these components are designed to work together to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system is run in a more or less predictable manner.

# Chapter 4

# Inside the soft real-time framework

This chapter begins with a description of our real-time process model and scheduling framework. Semantics of soft guarantee, real-time process categorization criteria, timing enforcement model, formats of real-time attributes and the basic scheduling framework are described. An examination of different real-time scheduling methods and their suitabilities to multimedia environments is also conducted. As a theoretical exploration, some general mathematical bounds are derived to show the performance upper bounds of soft real-time allowing loss.

## 4.1 Real-time process model and scheduling framework

Real-time OS for MM application should support a soft real-time programming model and a QoS-based resource management model. In the following discussions, we will consider the general application scenarios where CM application, non-CM applications are run competitively on a uniprocessor. We will also pay some attention to the case of communication subsystem where protocol processing is the main activity.

As mentioned before, the OS part of our soft real-time framework proposal consists of a process framework for categorizing processes, some timing enforcement models and some base scheduling and handling schemes. One of the overall goal in implementing our soft real-time framework is to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system is run in a more or less predictable manner.

## 4.1.1 Semantics of soft guarantee

We can differentiate between deterministic guarantee and probabilistic guarantee. For deterministic guarantee, all situations should be considered in the most pessimistic manner and the resource reservation and scheduling should be made in a very conservative manner so that a contracted guarantee can be maintained deterministically all the time. For probabilistic guarantee, in contrast, most situations can be viewed in a more optimistic manner and the resource reservation and scheduling can be made in a more opportunistic manner, i.e., less than peak resources need to be allocated. Instead, the statistical multiplexing of multiple applications and the felxibilities of these applications can be exploited to achieve a satisfactory probabilistic guarantee. Since conflicts between resource usages of different applications are possible, probabilistic guarantee means a contracted guarantee which might be broken from time to time.

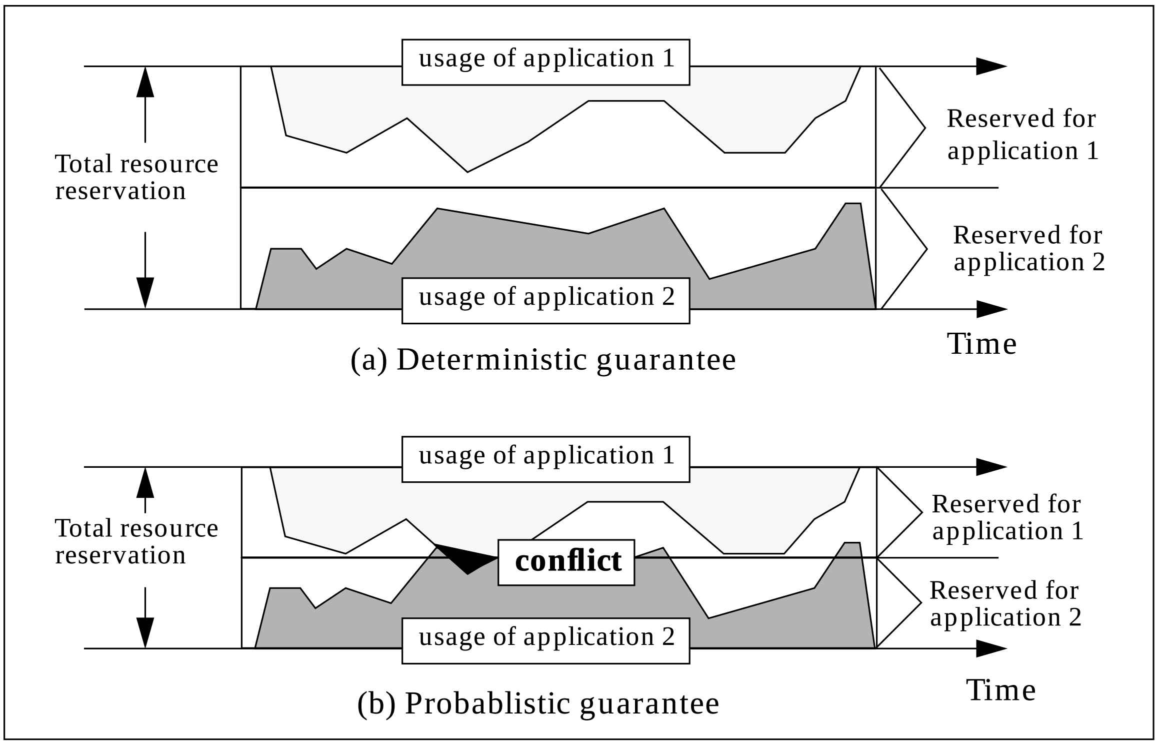
usage of application 1
Total resource reservation
Reserved for application 1
Reserved for application 2
usage of application 2
Time
(a) Deterministic guarantee
usage of application 1
Total resource reservation
conflict
Reserved for application 1
Reserved for application 2
usage of application 2
Time
(b) Probablistic guarantee

***Figure 4-1*** **Deterministic vs. probabilistic guarantee**

Figure 4-1 shows a case with two competitive applications.

In the context of multimedia, only soft, probabilistic guarantee is needed. In the following context, all the instances of “guarantee” indicate some forms of probabilistic guarantee.

In applying real-time scheduling algorithms in the multimedia context, two basic approaches can be taken to handle the occurrence of a broken soft guarantee.

The first approach is to treat the deadlines of real-time tasks as hard deadlines. After the expiration of a deadline, the real-time task will not be executed any more. The violation of soft guarantee appears then in the form that some real-time tasks can not be completed. In computing the success rate of the packet arrival before its playback-point, this interpretation will be applied.

The second approach is to treat the deadlines of real-time tasks as soft deadlines. The execution of a task after the expiration of its deadline has still some value to the system. In this approach, the violation of soft guarantee appears in the form that the deadlines are sometimes met in a degraded form. Such interpretation will be found in ET-SCHEDULE and adaptive service provision.

### 4.1.2 Real-time process model and real-time attributes

#### 4.1.2.1 Process categorization

We propose to use the following process framework. Processes in this framework will be treated in terms of processing possibility in three categories: “sure” (but not absolute) guarantee, “maybe” guarantee and best effort. The processes in the first two categories have to make an explicit claim about their timing constraints and will then receive a corresponding service — with “sure” and “maybe” soft guarantees of their timing constraints respectively. (Timing constraint specification will be detailed later and will contain such real-time (RT) features as deadline, period duration, worst-case execution time, etc.) The processes in the third category receive service when extra system capacity is available after the processes in the first two categories are served. Avoidance of starvation for the “best-effort” processes should be taken into account.

The rationale for the above categorization is that the most important MM activities can then be favored at most. Less important activities can be less favored. Other activities take what is left.

#### 4.1.2.2 Real-time attributes and timing overflow

There are several reasons to specify the real-time attributes of soft real-time processes. The specifications of the real-time attributes are part of the interfaces between the supporting system and the user processes. The real-time attributes describe the load an application will presumably place on the system. The system will partly base on these specifications to make its decisions on admission control and scheduling.

The concrete specifications of timing attributes depend on what kinds of processes the system will support directly, on the criteria of admitting a new process into a system, as well as on the scheduling strategies used by the system. On the one hand, it should be easy for the user to define and derive the attributes. On the other hand, it should be easy for the system to check, monitor and enforce the attributes.

Most continuous media-related activities are periodic in nature. Therefore, the periodic processes are directly supported in our framework. Two most important attributes of a periodic process is its period and the computation capacity requirement per period. Other attributes are necessary when more complicated management and scheduling strategies are used.

If accepted, the timing specification is then a kind of contract between the system and the user process. Both sides are assumed to try to behave according to the contract. In the case that a previously contracted timing specification is broken either by the system or by a user process, a timing violation or a timing overflow is said to have occurred.

#### 4.1.2.3 Timing enforcement models

There are two aspects to the enforcement of the above “timing constraint specifications”. First, the enforcement on the side of processes and the enforcement on the side of scheduling system are both relevant. That is, to what extent are the timing constraint specifications to be observed by both sides. Second, in the case of “violations” of timing constraint specifications of either side, what measurements should be taken and how these measurements should be taken. According to different timing enforcement models, different soft real-time programming models can be defined for different application needs.

From the viewpoint of the scheduling subsystem, the following three classes of timing enforcement models can be identified:

1) Cooperative. The scheduling system assumes that the timing constraints claimed by RT processes are also to be observed strictly by them. No monitoring or enforcement measurements are taken.

2) Imperative. The scheduling system enforces the timing constraint on the running processes. That is, the scheduler's view of period, deadline and execution time will be imposed on the running processes, regardless whether the running processes have really run in the claimed periods, with the claimed processing capacity, etc.

3) Semi-imperative (semi-cooperative). It is assumed that the timing constraints claimed by RT processes are usually observed by them. The scheduling system monitors their execution. In case of timing specification violation, some corrective measures may be taken. For example, a timing-violation-handler defined by the scheduling system or by the process itself may be called.

The choice of the models depends on the needs of the applications and the system environment, i.e., on the concrete semantics of the soft guarantee. They have different system effects. Note that the three models can be used in a system in a mixed way. The three models can be, for example, used for the above three categories of processes respectively or for other mixed usage.

In a system run in the cooperative or semi-imperative model, the processes themselves may also take some monitoring and enforcement measures to regulate themselves actively.

#### 4.1.2.4 Enforcement on periodic processes

Since periodicity is a typical feature of the MM activities, the following will try to deliberate the above enforcement models for the case of periodic processes.

A simple realization framework for a periodic process can be explained in the following way:

```
1
2      /* Initialize timing control variables. */
3      next_activation_time = start_time;
4      set_reincarnation_timeout(start_time);
5      loop {
6          sleep_until_reincarnation_timeout;
7          taskbody();             /* Do useful things here. */
8          next_activation_time = next_activation_time + task_period;
9          if (next_activation_time > end_time) then {break;}
10         else {set_reincarnation_timeout (next_activation_time);}
11     } /* end loop */
12
```

***Figure 4-2*** **Realization framework for a periodic process**

The periodic process is realized by an endless loop of its task body. The first instantiation of `sleep_until_reincarnation_timeout (line 6)` is waken up by `set_reincarnation_timeout(start_time) (line 2)`, thus the first instantiation of the task body is activated at `start_time`. The following instantiations of the task body are reincarnated repeatedly with a period of `task_period` by the repeated use of `set_reincarnation_timeout (line 10)` and `sleep_until_reincarnation-_timeout (line 6)` until `end_time (line 9)`.

With the cooperative periodic process model, the periodic processes independently use the above timing facilities to reflect and embody their own timing constraints. It is therefore assumed that each instantiation of the task body will be executed within `worst_case_execution_time` and the OS does not need to impose any extra constraints on its execution. The model is direct and simple. But it is then impossible to have a good control of timing violations caused by imprecise estimation of the processor usage (execution time) of the processes, by other transient overload conditions, or by synchronization conditions.

With a pure imperative periodic process model, the timing constraints of the processes should be enforced by the OS. Thus, a process is not allowed to use the above timing facilities directly and unconstrainedly. According to the real-time attributes claimed by the process, the OS decides implicitly when and for how long a process can be executed. Clearly, no timing violations on the side of processes can ever allow to happen, but the overhead of such enforcement is not small. Another main disadvantage here is that the intended natural timing features such as periodicity might not be the same as those actually enforced by the OS.

With a semi-imperative periodic process model, the periodic processes may still use the above timing facilities provided by the supporting OS to reflect and embody their own timing constraints or the periods may be activated by the supporting system directly. The OS monitors the execution of the process to detect possible timing violations. There are several varieties of timing violations and violation handling. If the execution of the task body has ever exceeded the `worst_case_execution_time` as claimed by the process in its real-time attributes, then it is a timing violation from the side of the process. If the OS detects that the system can not provide enough processing cycles to meet the need of the claimed timing requirements of the process, then it is a timing violation from the side of the OS. In the case of a timing violation, handling can be done by a OS procedure or a process-specific procedure. The method of handling can be roughly classified into abortive and corrective. The use of these varieties will be further discussed later in Chapter 5.

In our opinion, a semi-imperative process model is needed for most cases in most systems.

### 4.1.3 Basic scheduling framework

Our scheduling subsystem proposal contains an Admission Controller and a Soft Real-Time Dispatcher.

Since the capacities of the current computing systems are too limited to support many MM streams simultaneously, a satisfaction analysis should be conducted if the qualities of MM transmissions and processing are to be guaranteed to stay at a satisfactory level (for further and more general arguments, see Chapter 2). That is, new communication and processing requirements will usually not be accepted if the acceptance would violate the maintenance of the existing MM applications and communications. (But note again that both the “sure” guarantee and the “maybe” guarantee are probabilistic guarantees.)

As argued in Chapter 2, in order to achieve a certain degree of guarantee, resource reservations concerning CPU processing capacity, network bandwidth, buffer memory etc. should be done. It is the responsibility of the Admission Controller to check whether to allow the creation of a new process by considering the current load of the system and the real-time attributes of the incoming new process. The outcome of the decision of the Admission Controller can be one of the following three cases: a “sure” guarantee of the real-time attributes of the new process can be maintained; a “maybe” guarantee of the real-time attributes of the new process can be maintained; or a “best-effort” maintenance of the real-time attributes of the new process will be done. The process creator has the right to decide whether to accept the decision of the Admission Controller by continuing or aborting.

The Soft Real-Time Dispatcher exploits certain scheduling paradigm to schedule the real-time processes in the whole system.

Generally, an Admission Controller should contain a policy component as well as a resource accounting component. The policy component is more or less responsible for the administrative matters concerning admission control. It can take the forms such as administrative rules about user priorities, access preferences, real or virtual billing, etc. The resource accounting component is more or less responsible for the technical aspects of admission control. It deals with the technical possibilities of sharing the resources while maintaining service levels such as sure-guarantee and maybe-guarantee.

In the case of admission control for scheduling, the policy component deals with such preferences as which processes to preempt in the case of need. The resource accounting component computes the schedulability test proper according to the results of some scheduling theory.

#### 4.1.3.1 Guarantee possibility for CM and non-CM activities

It is important to point out that the categorization criteria of our framework are not directly MM-dependent. In our framework, the guarantees are not only provided for CM-related activities. Other non-CM-related activities can also receive guarantees if they are needed. (The framework by other people have quite often some limitations on this. Work such as [Wolf96] allows only CM-related processes to be favored.)

This important feature of our framework is especially important for composite MMC scenarios. For example, in the scenario of multimedia collaboration, a compilation job might have to deliver some results which should be diagnosed by the cooperating partners at different sites. Then it is important that this compilation job be guaranteed to proceed steadily while the other audio/video streams are delivered between partner sites. One way to achieve this is to serve the job as a virtual periodic process with sure guarantee so that the compilation process can be guaranteed to proceed quickly enough. (For an interface to virtual periodic process, please refer to *Section 5.4.1 Implications for the implementation schemes of applications* and *Section 7.2.2 Sample programming interface of usage*.)

#### 4.1.3.2 Admission controller and QoS manager

We note that the Admission Controller need not be a fully on-line function module. Generally, schedulability test of process sets has to be done with regard to CPU usage as well as other resource usage and real-time constraints. For complicated cases, a separate (off line) analyzer like the Scheduler-1-2-3 of the ARTS system [Tokuda89] is necessary in order to make an analysis of a large set of RT processes with complicated interdependency. It is then possible for the users to submit a set of processes to the Admission Controller together and the Admission Controller will then do a schedulability analysis and give its estimation result on their satisfiability.

Under many circumstances in MMC, a set of complimentary applications are delivered to the system together. Many of the applications might specify their QoS requirements in the form of requirement ranges such as {minimumQoS, desirableQoS, maximumQoS}. These QoS requirements correspond to the feasible ranges in resource requirements. In this case, a QoS management server can be used to balance the QoS requirements of various applications semi-automatically. The admission control is made with regard to the whole set of application processes. The QoS manager also plays an important role in supporting the FAST adaptive service model (see Chapter 9 on “Adaptive service provision”).

For periodic MM applications, each application specifies a {minimumTime, desirableTime, maximumTime} range of computation time required for a periodic interval. After a successful admission, each application is notified with the concrete computation time granted by the QoS manager. For normal MM application, it is feasible to implement a relatively simple Admission Controller based on the analysis of CPU utilization bounds and a small set of heuristics.

The algorithms used in checking schedulability and in scheduling should, of course, be compatible.

As will be further analyzed below, many of the existing real-time scheduling strategies can be used to support MM in our framework. The question is the degree of the ease of implementation and use, and the degree of flexibility of the corresponding real-time programming model. The possibility to extend these methods to support the soft RT features of MM applications should also be taken into account.

The choice of RT scheduling methods should, in the first place, be directed to the characteristics of the applications, be they event-driven or time-driven. Different scheduling possibilities can be used to support different types of MM processing models. It is therefore necessary to test the various scheduling strategies to see the combined system effects under different system environmental settings and conditions. In this sense, we second the ITDS [Tokuda89] idea of configurabiliy of scheduling and take a similar implementation approach. In the scheduling subsystem, a strategies/mechanisms separation method is adopted in order to make the implementations and comparisons easy. That is, a process scheduler is divided into strategy and mechanism modules. A strategy module may then embody a different scheduling algorithm. A mechanism module will then implement a set of routines for manipulating various ready, waiting and event queues. By carefully designing the interface between strategy modules and mechanism modules, flexible combinations of the strategy modules and a relatively stable mechanism module can be evaluated and compared.

## 4.2 Real-time scheduling methods and their suitabilities for MM applications

In this section, a brief overview of real-time scheduling methods is provided. We analyze their suitability for scheduling MM-related RT activities by comparing their relative advantages and disadvantages in the MM context. The purpose of this overview is to find scheduling elements that can be used directly in our soft real-time framework and to point out the possibilities of using some elements for other purposes.

### 4.2.1 An examination of scheduling methods

In the context of real-time scheduling, a range of different scheduling methods/ algorithms have been developed. The diversity of the algorithms can be seen in some survey papers such as [Audsley91, Cheng88, Mercer92].

Different scheduling algorithms are designed for different environments or for different application types. Some algorithms can be used for scheduling periodic tasks, some for aperiodic tasks. Some algorithms can be used on uniprocessor, some on a multiprocessor or in a distributed environment. Some scheduling algorithms are static and suitable for pre-runtime scheduling, some are dynamic and suitable for scheduling on the fly.

For scheduling on uniprocessor, scheduling algorithms range from static pre-computed schedule, deterministic scheduling to dynamic scheduling. Here, we are mainly concerned with the simple scheduling algorithms which do not have a large complexity both in schedulability test and in scheduling. For multimedia environment, we are of the opinion that it is infeasible to use complex dynamic scheduling heuristics on the fly, since the cost of the scheduling itself will then be too large. We are, however, also of the opinion that it is too inflexible to use a pure static pre-computed schedule method such as the cyclic executive method, since such a method is not able to handle a general multimedia system with both dynamic real-time and dynamic non-real-time task loads.

In the following, we look at the scheduling algorithms based on earliest-deadline-first, rate-monotonic and imprecise computation model more closely. In our opinion, these methods are simple yet flexible and can be readily used in our scheduling framework.

To simplify the following discussions, some notations are defined first. In this section, we use the term task and process interchangeably.

The timing constraints of a task are specified in terms of one or more of the following parameters:

(a) The arrival time A; (b) The ready time R; (c) The worst case execution time C; (d) The deadline D.

A periodic process set T has elements $\{\tau_1, \tau_2, ..., \tau_n\}$, where n is the cardinality of T. Each $\tau_i$ has the following characteristics:

$D_i$ — deadline (relative to the beginning of the period)

$T_i$ — Period

$C_i$ — computation time per period

We define a simple periodic process as one with $C_i \leq D_i = T_i$ (i.e. the process has computation time less than their deadlines, and the deadline is equal to their period).

#### 4.2.1.1 EDF-centered scheduling

Earliest-deadline-first (EDF) algorithm (also called earliest-deadline algorithm or earliest due-date scheduling algorithm) schedules the process with the closest deadline first. The earliest-deadline-first algorithm is a dynamic priority algorithm in the sense that the process with the current closest deadline is assigned the highest priority in the system and therefore executes.

In the context of queueing system, the earliest-deadline-first algorithm is optimal in the following sense: if any algorithm can schedule a particular task set without missing any deadlines, the earliest-deadline-first algorithm can as well. The sufficient and necessary schedulability constraint for a set of simple periodic tasks is given as

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le 1 \qquad \text{(E-4-1)}$$

Hence, 100% processor utilization is possible. (By the way, it has been shown that for arbitrary process set in which process timing constraints are relaxed to allow deadlines not equal to periods, formula (E-4-1) is necessary but not sufficient).

A closely-related scheduling algorithm to EDF is the least-laxity-first algorithm, where laxity is defined as the deadline minus remaining computation time. Although the least-laxity-first algorithm is optimum in the same sense as EDF, this algorithm is relatively impractical because of the possible “thrashing” problem [Audsley91].

Dynamic priority scheduling algorithms, among them earliest-deadline-first, typically have slightly greater scheduling overhead than fixed priority scheme such as rate-monotonic method. This is because the range of dynamic priorities is usually greater than the range of static priorities, and dynamic priorities must also be recalculated at each decision point whereas static priorities do not change and thus do not have to be recalculated.

The strong points of the EDF-based scheduling algorithm are: (1) It is optimal in many cases. For a simple task set, 100% utilization is possible. (2) It provides a simple and consistent framework for both periodic and aperiodic tasks.

One of the biggest problems with the EDF-based scheduling scheme is that it is a bad scheme under transient overload and it is quite difficult to make a good admission control without detailed knowledge of the full task set and their interactions. Ideally, under overload, a scheduling algorithm should be able to select the most important tasks and executes them while discarding less important tasks. However, EDF scheduling like most other dynamic priority algorithms does not encode high-level importance information in the priority; the priorities typically reflect low-level, dynamically changing timing characteristics.

Earlier studies have observed that in moderately-loaded real-time systems, using an earliest-deadline policy to schedule tasks results in the fewest missed deadlines. When the real-time system is overloaded, however, an earliest-deadline schedule performs much worse than most other policies. This is due to earliest-deadline policy giving the highest priority to tasks that are close to missing their deadlines.

EDF scheduling or its simple extensions are used in several multimedia systems. Examples can be found in [Anderson90a, Hagsand94, Nakajima91, Nakajima92].

#### 4.2.1.2 RM-centered scheduling

The rate-monotonic (RM) scheduling algorithm is a kind of capacity-based algorithm. Capacity-based algorithms have been developed with the goal to provide more flexibility while retaining the predictability of a fixed static scheduling method such as the cyclic executive method. In their pure form, capacity-based algorithms only require information about the amount of computation needed by a task set and the amount of computation available in the processing elements. This is in contrast to other deterministic scheduling algorithms which depend on exact timing information of tasks to make scheduling decisions. Because of this feature, capacity-based algorithms, such as rate-monotonic algorithm, are more flexible then a fixed static scheduling algorithm and, at the same time, are more efficient and easier to implement than a dynamic scheduling algorithms.

The basic rule of rate-monotonic scheduling is quite simple. Given a set of independent periodic tasks, the rate-monotonic scheduling algorithm gives a fixed priority to each task and assigns higher priorities to tasks with shorter periods (rate-monotonic priority assignment according to frequency). The task set is then scheduled using a fixed-priority-based, preemptive scheduler.

A task set is said to be *schedulable* if all its deadlines are met, that is, if every periodic task finishes its execution before the end of its period. Based on the following theorem [Liu73], we can predict whether a simple periodic task set scheduled in the rate-monotonic manner is schedulable by computing the utilization of the task set and then comparing that utilization to a *schedulable bound.*

***Theorem 4-1*** A set of n independent periodic tasks scheduled by the rate-monotonic algorithm will always meets its deadlines, for all task phasings, if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le n\left(2^{\frac{1}{n}} - 1\right) \qquad \text{(E-4-2)}$$

where $C_i$ and $T_i$ are the execution time and period of task $\tau_i$ respectively.

Theorem 4-1 offers a sufficient (worst-case) condition that characterizes the scheduling of the rate-monotonic algorithm for the case of a simple periodic task set.

***Table 4-1*** **Worst-case scheduling bounds as a function of number of tasks**

| Scheduling Bounds | |
|---|---|
| Number of Tasks | Utilization Bound |
| 1 | 1.0 |
| 2 | 0.828 |
| 3 | 0.779 |
| 4 | 0.756 |
| 5 | 0.743 |
| 10 | 0.718 |
| $\infty$ | ln 2 (0.693) |

By way of period-transformation [Sha86], a long period task can be transformed into a shorter period task. If all the periodic tasks are transformed in the way such that the more important tasks have shorter periods, then the system under rate monotonic scheduling has the nice feature of graceful degradation under transient overload — the less important tasks (the tasks with longer periods) will miss their deadlines first.

A problem with RM scheduling is its utilization. For a system with many tasks, a pessimistic bound is about 69% (see Table 4-1). But this bound is pessimistic. In most cases, 88% is a more realistic bound [Lehoczky89]. In fact, the reservation bound is 100% for the case where all periods are harmonic, i.e., each periods is an even multiple of every periods of smaller durations. Additionally, an amount of unreserved scheduling time of perhaps 5 - 10% is usually necessary to avoid scheduling failures due to inaccuracy in the computation time measurement and enforcement strategy and due to the effects of critical regions and other synchronization and communication among tasks.

Around the basic rate-monotonic algorithms, many extensions have been made. The extensions under the name “Generalized Rate-Monotonic Scheduling Theory (GRMT)” can provide a fairly complete framework for developing real-time systems [Sha89, Sha94]. GRMT can both schedule periodic and aperiodic tasks with synchronization requirements and mode change requirements. GRMT with a set of theoretical results lays a sound foundations for RM-centered scheduling systems. This point will be further refined in *Section 5.1 The choice of GRMT as a cornerstone for the schemes*.

The use of rate-monotonic scheduling can be found in the ARTS and Real-time Mach systems [Tokuda89, Tokuda90, Tokuda92]. The latter has been used in several multimedia systems recently [Kihara93, Tokuda93].

#### 4.2.1.3 Imprecise scheduling

The imprecise computation model [Liu91, Chung90] argues that many computations are incremental in nature. It argues that it is useful to make available an acceptable but imprecise results in time, if the desired precise results can not be produced in time. Three methods of imprecise computations have been proposed: milestone, sieve and multiple version method.

The algorithms given in [Chung90] belong to milestone methods. Specifically, in the imprecise computation, real-time processes are designed to be monotone, that is, the accuracy of its intermediate results is nondecreasing as more time is spent to produce the result. If the intermediate result is usable, it is said to be acceptable. The imprecise computation approach makes scheduling real-time tasks significantly easier. To guarantee that all deadlines are met requires only that sufficient processor time be assigned to every task for it to produce an acceptable result before its deadline. The approach taken is to consider each task as consisting of two parts: a mandatory part, that must be completed in order for the task to produce an acceptable result, and an optional part, that refines the result produced by the mandatory part to reduce the error in the result.

In these algorithms, the mandatory parts of all tasks are assigned hard deadlines. The rate-monotonic algorithm is used to schedule them to meet all deadlines. The optional parts have soft deadlines; different algorithms are used to schedule the optional parts to minimize the average error.

In the sieve method, certain parts of the computation are considered optional and can be omitted. In multiple version method, a primary version of the task computes the precise result and an alternative version computes an imprecise (but adequate) result in a shorter period of time. The scheduler chooses between these two versions based on the runtime situation.

The idea and algorithms of the imprecise computation approach are surely also applicable to MM scheduling in some cases. In contrast to EDF-centered or RM-centered algoritms, however, the imprecise computation approach is more readily applicable as an application-level adaptive method.

#### 4.2.1.4 Other scheduling strategies

We would like to mention here the value-function scheduling method and various off-line / on-line dynamic scheduling heuristics.

The value-function scheduling algorithms use value functions associated with the tasks to make scheduling decisions. The main problem is that it is usually not easy to find feasible value function for dynamic systems and the computation complexity is high. Usage example in MM system can be found in [Wall92].

There are many off-line / on-line dynamic scheduling heuristics [Casavant88, Ramamritham89]. Their main problem is that they are usually too complex and too time-consuming to be used at low-level scheduling.

### 4.2.2 Criteria of a suitable scheduling scheme

As mentioned above, in applying the current scheduling algorithms in multimedia context, two basic approaches can be taken for two interpretations of soft guarantee. In both cases, the estimation of the real-time attributes of the real-time tasks in multimedia environment can be done in a more optimistic manner, because hard guarantee is not a necessity.

Because most of the existing scheduling algorithms are for rigid real-time systems, they generally impose some strict conditions on the task set. In the practice of MM systems, some of the conditions may not be met all the time. For example, real-time processes might have a more complicated interdependency as assumed, they might not be completely preemptive at all points, context switch times might vary, etc. Thus, the systems will not adhere to the reservations as strictly as the scheduling models predict. But, with the aid of accurate execution monitoring and by exploiting the flexibility of MM applications, the transient overload and timing violations can be detected and corrective actions can be taken both at a low-level on a fine-gram time scale and at a high-level on a coarser system time scale (adaptive service). Generally speaking, we can take a more relaxed approach to constructing MM system software than to constructing hard real-time system software with regard to scheduling.

For a scheduling method to be usable in our scheduling framework, the following prerequisites should be fulfilled:

1) it should provide the possibility to express timing constraints of processes;
2) it should provide the possibility to check the schedulability of a set of processes in order to do admission control;
3) optionally, it should provide the ability to control the processes hierarchically in order to let some of the processes to be “favored” over others.

Many real-time scheduling algorithms fulfill the above prerequisites and can, in this sense, be used in the MM environment.

According to these criteria, EDF-centered scheduling methods and RM-centered scheduling methods are apparently the most promising candidates. Other scheduling elements can also be incorporated to produce different scheduling strategies for different environments. For MMC scenario, we have designed a set of scheduling and handling schemes based on the Generalized Rate Monotonic Theory (GRMT). Experimentations and evaluations are conducted with regard to these schemes. *PART III Soft Real-Time Scheduling and Handling* presents the related results.

## 4.3 Mathematical bounds of soft real-time with loss

In a dynamic system, the real-time processes come and go dynamically. In this section, we consider a simplified model for a soft real-time system allowing loss. In such a soft real-time system, the system should try to service the real-time processes within their deadlines. If the deadline of a soft real-time process expires before it can be serviced by the system, the soft real-time process is considered to be useless or be lost. It will then be cleared and removed out of the system.

Due to the statistical fluctuation of the process arrivals and the limited capacity of the CPU, there are always the cases where even an optimal scheduling strategy is not able to arrange to service all the processes to meet their deadlines. This leads to a fundamental problem underlying the design and implementation of the scheduling strategies for scheduling soft real-time processes — a certain percentage of loss is inevitable for many situations.

In this section, the problem is considered under different assumptions about the deadline and computation time distributions of the arriving processes. These case are modeled by means of queueing models to get their theoretical performance limits.

### 4.3.1 Modelling soft real-time processing with loss

Define deadline D as the time before which a real-time process should be serviced. Define the computation time needed for the process as C. Let us further assume that the service of a process will be conducted to the end once the service has begun, i.e., non-preemptive. Define laxity L as the time between the creation time $T_c$ of a RT process and the latest time $T_L$ at which the service should begin. If the service begins after the latest beginning time $T_L$, then the deadline of the process will surely be violated. Therefore it doesn't make sense to begin the service after $T_L$. This can be seen more clearly in Figure 4-3. And we have the following relations:

$$T_L = D - C$$

$$L = T_L - T_c = D - C - T_c$$

And for any successful service: $C = S_e - S_b$.

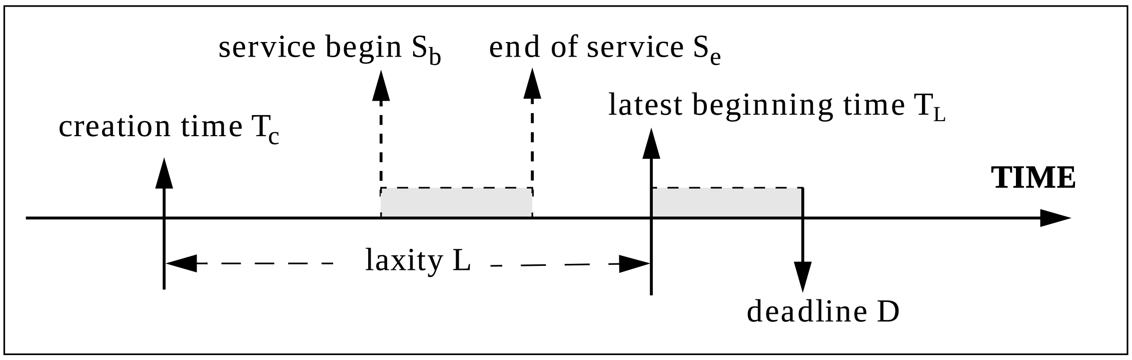

***Figure 4-3*** **Time constraint parameters**

In the queueing theory, the above situation is treated under the queueing problems with impatient customers — impatient customers will leave a waiting queue without being serviced [Baccelli84]. It has been shown that the system performance indices such as the utilization of the server and the rate of served customers can be stated in terms of the normalized offered load and the normalized mean laxity (normalized by the service rate).

We use an abstract queueing model for soft real-time system with loss as shown in Figure 4-4. The process arrivals are placed in the waiting queue for service. If the laxity of a process has expired, it will be thrown away from the queue and will be regarded as a loss.

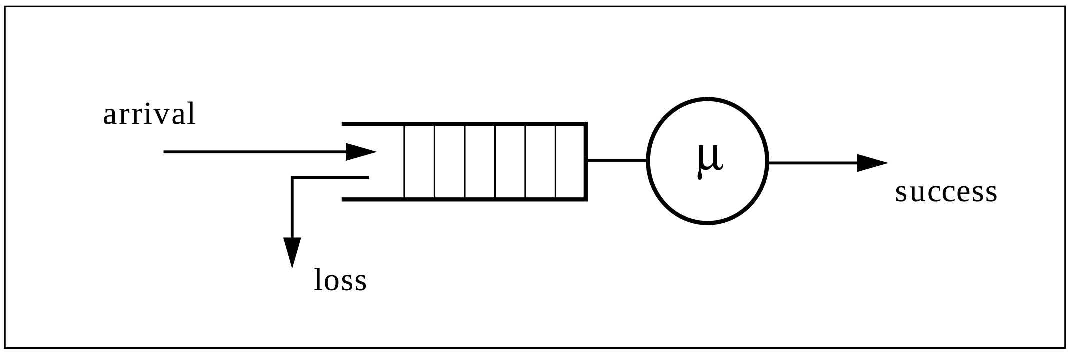

***Figure 4-4*** **Abstract queueing model of soft real-time system with loss**

In the following analysis, the mean process computation time (required service time) is normalized to 1. Other parameters are normalized with regard to the mean process computation time. The total arrival rate of processes normalized by the mean computation time is denoted as $\rho$ and the mean laxity of the process normalized by the mean computation time is denoted as ***b***. If, for example, the mean computation time of all processes is 15 time units, the arrival rate per time unit is 0.03 and the mean laxity is 75 time units. With the mean computation time being normalized to 1, we have $\rho$=0.03*15=0.45 and b=75 / 15=5.

- **Backlog of waiting processes**

For a non-real-time system, the processes will wait until it is processed / serviced in the end. Therefore, a non-real-time system is a work-conserving system. If the normalized arrival rate is less or less than 1, than the server (CPU) will be able to process all the processes in the end. The backlog of the waiting processes in the system, though, can be quite large from time to time due to arrival bursts. If the normalized arrival rate is larger than 1, than the server will be saturated eventually and the backlog will grow unlimitedly unless the arrival rate is tuned down to less than 1.

In the soft real-time system allowing loss as modeled by the above abstract model, the backlog of the processes in the waiting queue is practically bounded. This is due to the fact that the processes can wait at most L time long (with a mean of b) before they are either successfully serviced or become useless and thus be deleted from the waiting queue. In other words, the system is a non-work-conserving system. Recall Little's result [Kleinrock75]:

$$N = \lambda W \qquad \text{(E-4-3)}$$

where $\lambda$ denotes arrival rate, W denotes the average waiting time of customers in the queueing system (including service time), and $N$ is the average number of customers in the queueing system (in the queue or in service). The Little's result is general, without regarding the distribution of arrival, service and queueing strategy. Therefore we can get a bound of the mean number of backlogged processes as:

$$MeanBacklogBound = \rho b \qquad \text{(E-4-4)}$$

For $\rho$=0.7, b=10, for example, MeanBacklogBound= 7. This means, there are at average no more than 7 processes waiting.

Further more, the backlog caused by the short-term overloading will not have a long-term effect on the system functioning. This is due to the fact that the backlogged old processes will not exist in the system after L time long — either successfully serviced or lost. This implies that the stability problem is not a question in this context.

- **Upper bound of performance**

As stated above, due to the statistical fluctuation of the process arrivals and the limited capacity of the CPU, there are always the cases where even an optimal scheduling strategy is not able to arrange to service all the processes to meet their deadlines at all times. It is our intention to derive the theoretical performance limit of a single CPU in servicing soft real-time processes with loss. As such, we assume the cost of scheduling as zero.

The use of the derived theoretical performance upper bounds is twofold — They provide a guideline for the scheduling scheme designers and implementors by pointing out the optimal performance ever possible. They also provides a guideline for the scheduling scheme users in that the users would be aware of the limits of a scheduling scheme under extreme conditions, even if the scheduling scheme in question does provide acceptable performance under normal operation.

Note that the disappeared, previously backlogged processes are actually lost, if not successfully serviced. The above phenomenon of low backlog actually means that the many incoming processes in a bursty period have to be serviced during the same period even if the process arrival rate in its preceding or following period is very load. At the same time, the steady-state performance indices for a high arrival rate are also useful for a system with a low mean arrival rate — in its peak periods, the short-term system performances are approximate to the case of lasting high load. It is therefore meaningful to know the scheduling performance under all different loads.

Specifically, a main performance index is the percentage of process loss under different system conditions. We are interested in the performance indices for the service (scheduling) schemes which are “fair” with regard to service time (computation time) requirements. (The “unfair” service schemes which favor processes with short service time requirements will achieve a higher success rate. But this unfairness is not feasible for most system circumstances.)

Earliest-deadline-first (EDF), least-laxity-first (LLF) and first-come-first-serve (FCFS) scheduling/service schemes are all fair schemes which have no bias with regard to service time requirements. As stated in *Section 4.2 Real-time scheduling methods and their suitabilities for MM applications*, the EDF and LLF schemes are optimal in the following sense: if any algorithm can schedule a particular process set without missing any deadlines, then the earliest-deadline-first algorithm and the least-laxity-first algorithm can as well. Since it is sometimes mathematically intractable to model a dynamic priority scheduling algorithms such as LLF or EDF, we will then make do with a FCFS analysis.

In all the models below, the total arrival rate of the system is assumed to be a Poisson process, i.e., the inter-arrival time of the processes are exponentially distributed. The models are named by an extended form of Kendall’s notation A/B/m+L [Zhao89]. ***A*** describes the inter-arrival process, ***B*** describes the service time requirements, ***m*** is the number of servers, and ***L*** represents the distribution of customer laxity. In our case, A is always M, m is always 1, and L corresponds to the process laxity distribution.

### 4.3.2 M/M/1+D model

Using the M/M/1+D model, we model the case where all the incoming processes have the same laxity to their deadlines. In this model, a least-laxity-first service scheme (LLF) takes the same form as the first-come-first-serve service scheme (FCFS). In the following, we try to derive the upper bound of the success percentage of the processes which can be serviced by the LLF/FCFS scheme.

Define F(w,t) as the *pdf* of the unfinished work (unserved waiting processes) in the queue at time *t*. Following the approach in [Baccelli84, Kurose87, Zhao89], the pdf of the unfinished work at time t+Δt for M/M/1+G can be characterized as follows:

$$F(w, t+\Delta t) = (1-\lambda\Delta t)F(w+\Delta t, t) + \lambda\Delta t\int_0^w (1-L(x))B(w-x)dxF(x,t) + \lambda\Delta t\int_0^w L(x)dxF(x,t) \qquad \text{(E-4-5)}$$

where L(x) is the process laxity distribution and B(x) is the service time distribution. In the above equation, the left hand side is the probability that at time t+Δt, the total unfinished work is less than w. On the right hand side, the first term is for the case where there is no new arrival from time t to t+Δt. The second term is for the case where there is a new arrival and the new arrival can be serviced. The last term is for the case where new arrival can not meet its deadline.

Let F(w) denote the steady-state solution as t —> ∞. For the case of M/M/1+D where the time laxity is a constant K, the loss rate can be given as [Kurose87, Kurose88]:

$$\lambda_{loss} = \lambda(1-F(K)) \qquad \text{(E-4-6)}$$

For our use, the percentage of success P can then be obtained as:

$$P = \frac{\lambda - \lambda_{loss}}{\lambda} = F(K) \qquad \text{(E-4-7)}$$

The following tries to get an explicit expression of P for the current case. In [Kurose87], the following steady-state equation has been obtained for the case that the service time is exponentially distributed with a mean 1/μ:

$$F(w) = F(0^+)\left(\frac{\mu}{\mu-\lambda} - \frac{\lambda}{\mu-\lambda}e^{-w(\mu-\lambda)}\right) \qquad (0 < w \le K) \tag{E-4-8}$$

Let w=K, it follows:

$$F(K) = F(0^+)\left(\frac{\mu}{\mu-\lambda} - \frac{\lambda}{\mu-\lambda}e^{-K(\mu-\lambda)}\right) \tag{E-4-9}$$

Note we also have the following flow conservation equation, which is based on the fact that all the arrival will either be successfully served or be lost:

$$\lambda = (1 - F(0^+))\mu + (1 - F(K))\lambda \tag{E-4-10}$$

By combining equations (E-4-9) and (E-4-10), we get the following solution:

$$F(K) = \frac{\mu^2 - \mu\lambda e^{-K(\mu-\lambda)}}{\mu^2 - \lambda^2 e^{-K(\mu-\lambda)}} \tag{E-4-11}$$

For λ, K being normalized to μ and μ being normalized to 1, we get the following result by combining equations (E-4-7) and (E-4-11):

$$P = \frac{1 - \rho e^{-b(1-\rho)}}{1 - \rho^2 e^{-b(1-\rho)}} \tag{E-4-12}$$

where ρ and b are normalized arrival rate and normalized mean laxity respectively.

Some numerical results of the equation (E-4-12) are given in Table 4-2 and the corresponding curves are presented in Figure 4-5 and Figure 4-6.

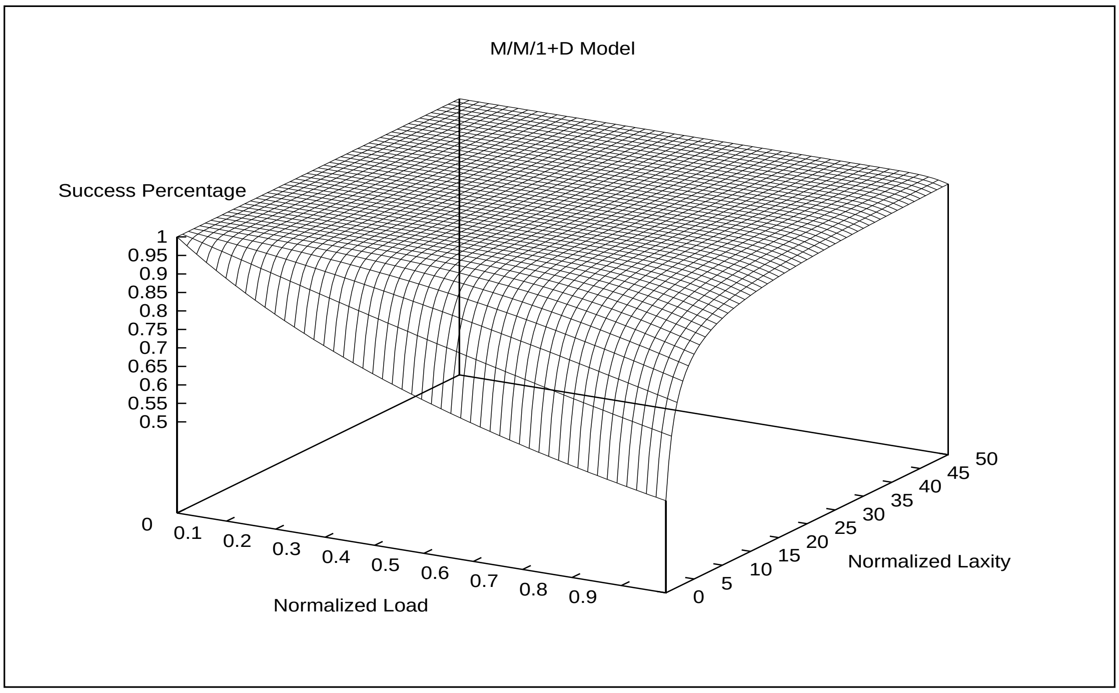

***Figure 4-5*** **Success percentage for M/M/1+D model**

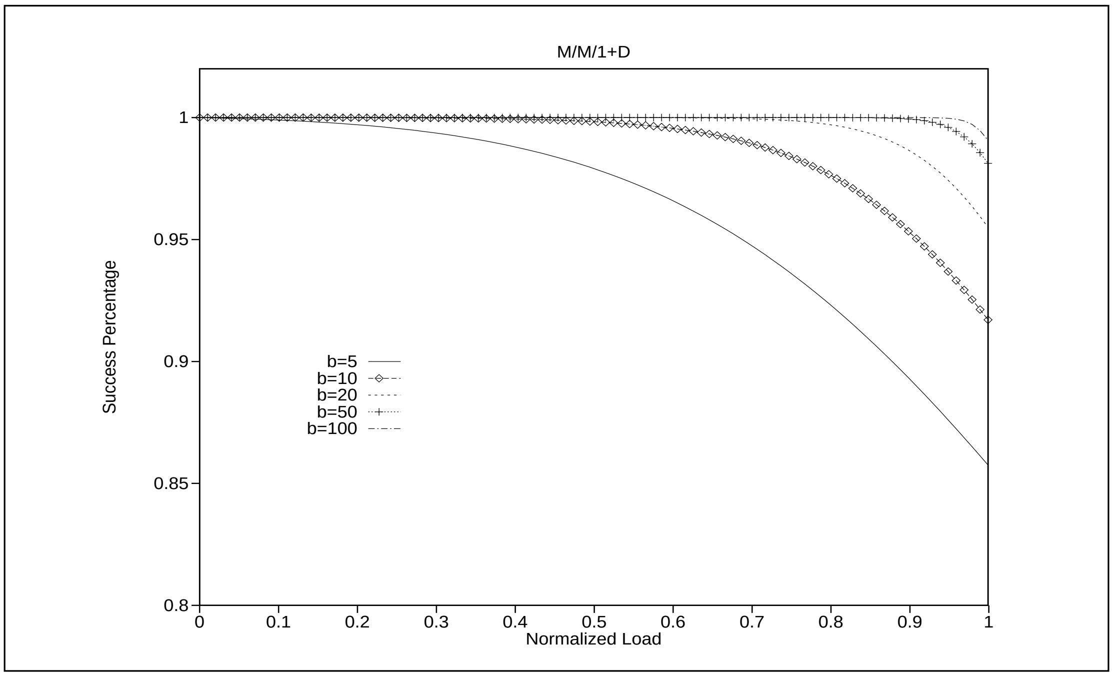

***Figure 4-6*** **Sample success percentage for M/M/1+D model**

***Table 4-2*** **Success percentage from formula (E-4-12)**

(* P is always less than 1. A value of 1.0 in the following table indicates a value larger than 0.9999.)

| $\rho$ | 0.3 | 0.5 | 0.7 | 0.9 | $1.0^-$ |
|---|---|---|---|---|---|
| **P** (b=5) | 0.9936 | 0.9790 | 0.9474 | 0.8927 | 0.8571 |
| **P** (b=10) | 0.9998 | 0.9983 | 0.9893 | 0.9528 | 0.9167 |
| **P** (b=20) | 1.0* | 1.0* | 0.9995 | 0.9863 | 0.9545 |
| **P** (b=50) | 1.0* | 1.0* | 1.0* | 0.9994 | 0.9808 |
| **P** (b=100) | 1.0* | 1.0* | 1.0* | 1.0* | 0.9902 |

### 4.3.3 M/M/1+M model

For a M/M/1+M model with FCFS service, the probability of process loss has been derived as [Zhao89]:

$$R = R^+(1 - F(0^+)) \tag{E-4-13}$$

where

$$R^+ = 1 - \frac{\gamma(b+1, \rho b)}{\rho b \gamma(b, \rho b)} \tag{E-4-14}$$

$$F(0^+) = \frac{1 - \frac{\gamma(b+1, \rho b)}{b\gamma(b, \rho b)}}{1 + \left(\rho - \frac{\gamma(b+1, \rho b)}{b\gamma(b, \rho b)}\right)} \tag{E-4-15}$$

and $\gamma(a, x)$ denotes the incomplete gamma function

$$\gamma(a, x) = \int_0^x y^{a-1} e^{-y} dy \tag{E-4-16}$$

It follows that the success percentage of process service is:

$$\begin{aligned} P &= 1 - R \\ &= 1 - \left(1 - \frac{\gamma(b+1, \rho b)}{\rho b \gamma(b, \rho b)}\right)\left(1 - \frac{1 - \frac{\gamma(b+1, \rho b)}{b\gamma(b, \rho b)}}{1 + \left(\rho - \frac{\gamma(b+1, \rho b)}{b\gamma(b, \rho b)}\right)}\right) \end{aligned} \qquad \text{(E-4-17)}$$

Some numerical results from equation (E-4-17) can be found in Table 4-3.

***Table 4-3*** **Success percentage for M/M/1+M**

| ρ | 0.3 | 0.7 | 0.9 | $1.0^-$ |
|---|---|---|---|---|
| **P** (b=10) | 0.9663 | 0.8938 | 0.8422 | 0.8119 |
| **P** (b=100) | 0.9958 | 0.9803 | 0.9548 | 0.9293 |

## 4.3.4 Other models

As can be seen from the above analysis, it is usually quite complicated to derive closed-form solutions for A / B / m+L models. For more complicated distributions as A, B, or L, it is virtually mathematically intractable. In [Fan92], we have proposed to model and evaluate time-constrained message transmissions by Generalized Stochastic Petri Nets (GSPN). This approach has the advantages of explicitness, flexibility and ease of use. With the help of some GSPN tools, it is possible to model and evaluate some quite complicated and composite scenarios. For example, the modeling of time-constrained message transmission over a single channel has been conducted under different assumptions. This result can be readily used in our current case where the CPU corresponds to the central channel. Some numerical results of different models on FCFS service scheme are compared in Table 4-4.

***Table 4-4*** **Success percentage comparison of different models**

| Model | M/M/1+M | M/D/1+M | M/Mx/1+Mx | M/Er/1+Er | M/M/1+D | M/D/1+D |
|---|---|---|---|---|---|---|
| **P** (b=10) | 0.8119 | 0.8511 | 0.8700 | 0.9137 | 0.9165 | 0.9539 |

## 4.4 Summary of the chapter

The OS part of our soft real-time framework proposal consists of a process framework for categorizing processes, some timing enforcement models and some base scheduling and handling schemes. We propose to use the following process framework. Processes in this framework are treated in terms of processing possibility in three categories: "sure" (but not absolute) guarantee, "maybe" guarantee and "best-effort". The processes in the first two categories have to make an explicit claim about their timing constraints and, if accepted by the admission control of the scheduling subsystem, will then receive a corresponding service — with "sure" and "maybe" guarantees of their timing constraints respectively. In our framework, it is possible to provide guarantee for both CM and non-CM-related activities. We consider the following three timing enforcement models: (1) cooperative; (2) imperative; (3) semi-imperative. The usage of the models depends on the needs of the applications and the system environment, i.e., on the concrete semantics of the soft guarantee.

Our scheduling subsystem proposal contains an Admission Controller and a Soft Real-Time Dispatcher. It is the responsibility of the Admission Controller to check whether to allow the creation of a new process by considering the current load of the system and the real-time attributes of the incoming new process. The Soft Real-Time Dispatcher exploits certain scheduling paradigm to schedule the real-time processes in the whole system.

For a scheduling method to be usable in our scheduling framework, the following prerequisites should be fulfilled: (1) possibility to express timing constraints of processes; (2) possibility to check the schedulability of a set of processes in order to do admission control; (3) (optionally,) ability to control the processes hierarchically in order to let some of the processes to be “favored” over others. After an examination of the currently available scheduling algorithms, we conclude that the EDF-centered scheduling methods and RM-centered scheduling methods are apparently the most promising candidates. Other scheduling elements can also be incorporated to produce different scheduling strategies for different environments.

In a dynamic soft real-time system, due to the statistical fluctuation of the process arrivals and the limited capacity of the CPU, there are always the cases where even an optimal scheduling strategy is not able to arrange to service all the processes to meet their deadlines. This leads to a fundamental problem underlying the design and implementation of the scheduling strategies for scheduling soft real-time processes — a certain percentage of loss is inevitable for many situations. We consider the problem under different assumptions about the deadline and computation time distributions of the arriving processes. Some case are modeled by means of queueing models. Their theoretical performance bounds are derived and some numerical examples are presented.

# PART III

# Soft Real-Time Scheduling and Handling

# Chapter 5

# Design, realization and usage of soft real-time scheduling and handling schemes

Soft real-time scheduling and handling schemes are essential for the realization of a soft real-time framework. As the first chapter of Part III, this chapter describes a set of scheduling and handling schemes which can be used in a concrete multimedia system. The simulation model and the simulation evaluation results of these schemes are then presented in Chapter 6. The experimental implementation of these schemes on a hardware platform is then documented and the related measurement results are analyzed in Chapter 7.

This chapter begins by stating the reasons why we choose the Generalized Rate Monotonic Theory (GRMT) as the cornerstone of our scheduling and handling schemes. It then outlines the concrete design of our soft real-time scheduling mechanisms and soft real-time handling methods. The overall realization issues of the soft real-time framework with these schemes are then discussed. Feasible alternatives for implementations are compared. Usage schemes for application implementations and other practical issues are also explored.

## 5.1 The choice of GRMT as a cornerstone for the schemes

As will be detailed in the next section, our scheduling methods are designed by integrating some elements from the rate-monotonic scheduling, the priority-based scheduling and the weighted round-robin scheduling. The Generalized Rate Monotonic Theory [9] is used in all the methods as a basis for admission control and for real-time scheduling. The other related results of the theory are also used in other aspects of the system implementation such as preventing unbounded priority inversion by some forms of priority inheritance.

The choice of the generalized rate monotonic scheduling theory as a cornerstone of our scheduling methods is based on two main reasons: the completeness of the GRMT theory for many real-time environments, and the practical advantages of the GRMT-oriented real-time schemes over many other real-time methods.

### 5.1.1 Broad coverage of GRMT

The basic form of a rate monotonic scheduling algorithm which can only be applied to a set of simple periodic task set has been introduced in *Section 4.2.1.2 RM-centered scheduling*. The generalized rate monotonic scheduling theory [Sha90, Klein93, Sha94] has made many extensions to this basic form.

For sake of clarity of the following design and analysis, the following cites and explains some of the most-related results of the GRMT here.

To determine if tasks scheduled on a resource with utilization greater than the bound of Theorem 4-1 can meet their deadlines, the following exact schedulability test based on the critical zone theories can be used:

***Theorem 5-1*** For a set of independent periodic tasks, if a task $\tau_i$ meets its first deadline $D_i \leq T_i$, when all the higher priority tasks are started at the same time, then it meets all its future deadlines with any other task start times.

It is important to note that the above Theorem 5-1 applies to **any** static priority assignment, not just rate-monotonic priority assignment. This insight is very useful in analyzing the schedulability for the case where some higher priorities must be given to some special tasks not in the strict rate-monotonic order.

The checking of Theorem 5-1 can be represented by an equivalent mathematical test:

***Theorem 5-2*** A set of n independent periodic tasks scheduled by the rate-monotonic algorithm will always meets its deadlines, for all task phasings, if and only if

$$\forall i, 1 \le i \le n, \min_{(k,l) \in R_i} \sum_{j=1}^{i} C_j \frac{1}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \le 1 \qquad \text{(E-5-1)}$$

where $C_j$ and $T_j$ are the execution time and period of task $\tau_j$ respectively and $R_i = \{(k, l) \mid 1 \le k \le i, l = 1, ..., \lfloor T_i / T_k \rfloor \}$.

Tasks are more often dependent on each other. They interact by sharing and mutual-excluding. It is then necessary to use synchronization primitives such as semaphores, locks, monitors and Ada rendezvous. A direct application of these synchronization mechanisms may lead to an indefinite period of *priority inversion*, which occurs when a high-priority task is prevented from executing by a low-priority task. To tackle the problem of priority inversion, the GRMT provides a real-time synchronization protocol called priority ceiling protocol.

There are two main parts of the design of the priority ceiling protocol. The first is the concept of priority inheritance: when a task $\tau$ blocks the execution of higher priority tasks, task $\tau$ executes at the highest priority level of all the tasks blocked by $\tau$. The second is the use of priority ceiling to enforce a prioritized total ordering. The priority ceiling of a binary semaphore S is defined to be the highest priority of all tasks that may lock S. When a task $\tau$ attempts to execute one of its critical zones, it will be suspended unless its priority is higher than the priority ceilings of all semaphores currently locked by tasks other than $\tau$.

The priority ceiling protocol has two important properties:

1) freedom from mutual deadlock and
2) bounded priority inversion. Namely, a high-priority task will be blocked by at most one critical region of any lower-priority task.

Let $B_i$ be the longest duration of blocking that can be experienced by task $\tau_i$. The following theorem can be used to determine whether the deadlines of a set of periodic tasks can be met if the priority ceiling protocol is used:

***Theorem 5-3*** A set of n periodic tasks using the priority ceiling protocol can be scheduled by the rate monotonic algorithm for all task phasings, if the following condition is satisfied:

$$\sum_{i=1}^{n}\frac{C_i}{T_i}+max\left(\frac{B_1}{T_1},\ \ldots,\ \frac{B_{n-1}}{T_{n-1}}\right)\le n\left(2^{\frac{1}{n}}-1\right) \qquad \text{(E-5-2)}$$

This theorem generalizes Theorem 4-1 by taking blocking into consideration. A similar extension can be made for Theorem 5-2. The $B_i$'s in Theorem 5-3 can be used to account for any delay caused by resource sharing.

In addition to the above-cited results, the generalized rate monotonic scheduling theory has extended the basic form of RM scheduling from its original form of scheduling independent periodic tasks to scheduling

- both periodic and aperiodic tasks (polling server, sporadic server),
- tasks with synchronization requirements (priority-inheritance, priority-ceiling),
- tasks with deadlines before the end of the period (deadline-monotonic),
- tasks with mode change requirements,
- tasks with deadlines after the end of the period.

In addition, the following issues have also been addressed:

- precise algorithms for determining schedulability,
- associated hardware scheduling support,
- implications for Ada scheduling rules and algorithm implementation in an Ada runtime system,
- schedulability analysis of input/output paradigms.

Furthermore, numerous design and analysis experiments have been performed to test the viability of the theory. Together, these findings constitute a rich set of analytical and realization methods for real-time system engineering.

### 5.1.2 Practical advantages

The broad coverage of the GRMT theory means that it can provide a sound theoretical basis for the handling of even the most complicated application scenarios. It would also be possible for us to apply the same set of schemes based on GRMT to many different applications and environments. So the first biggest advantage of the use of GRMT lies in its completeness and soundness of the theory and its broad coverage of applicability.

- **Utilization bound not a problem**

One of the most-cited arguments against the use of GRMT for real-time is the relatively-low worst usage bound. The worst case utilization bound for the basic form of rate monotonic scheduling was shown by Theorem 4-1 to be $n(2^{(1/n)}-1)$, a quantity which decreases monotonically from 0.83 when n=2 to ln2=0.693 as n—>∞. This bound is actually very pessimistic because the worst-case task set is contrived and unlikely to be encountered in practice. The average case behavior is substantially better than the worst case behavior. In common practice, the rate monotonic algorithm can usually successfully schedule task set having total utilization higher than 69.3%. Indeed, it is not uncommon for large periodic task sets with total utilization bound larger than 90% to be schedulable if an exact analysis is conducted by using Theorem 5-2 or other techniques. For a randomly chosen task set, the likely bound is 88% [Lehoczky89]. The reservation bound of rate monotonic is 100% for the special case where all periods are harmonic, i.e. each period is an even multiple of every period of smaller duration.

In our context of soft real-time, this seemingly-low utilization bound is still less a problem. The computation capacity for each soft real-time application can be estimated in a more optimistic manner. I.e. less than worst case should be reserved. This is especially true for the applications which need “may-be” guarantee only. Additionally, an amount of unreserved computation time of at least 5 to 10% is generally necessary to avoid the starvation of the tasks in the best-effort category. Such an amount of unreserved spare computation capacity may also be very helpful in avoiding scheduling failures due to inaccuracy in the computation time measurement and enforcement mechanisms and due to the effects of critical regions and other synchronization and communication dependency among tasks. With a view to the rapid development in hardware, it is also far more important to have a theoretically-sound predictability than to simply strive for a higher utilization.

- **GRMT vs. EDF etc.**

Compared to some complicated dynamic scheduling heuristics, the choice of rate monotonic scheduling as a cornerstone of our scheduling methods lies also in its simplicity and flexibility. Although earliest-deadline-first (EDF) and related scheduling algorithms have the similar advantages with regard to simplicity and flexibility, they are inferior to GRMT in several aspects.

Although techniques such as kernelized monitor can be used for synchronization by EDF, there does not exist as complete a set of EDF-related high-utilization theorems for complicated scenarios as GRMT.

A typical phenomenon that may happen with EDF when the system is overloaded is the “domino effect”, since a first task that misses its deadline may cause many subsequent tasks to miss their deadlines. In such a situation, EDF does not provide any type of guarantee on which tasks will meet their timing constraints. This is a very undesirable behavior in practical systems, since in real-world applications intermittent (transient) overloads may occur due to exceptional situations, such as modifications in the environment, arrival of burst of tasks, or cascades of system failures. Under cooperative scheduling model, the computation time should then be predicted either exactly or very pessimistically to avoid such phenomena.

Dynamic priority scheduling algorithms, among them earliest-deadline-first, typically have slightly greater scheduling overhead than fixed priority scheme such as rate-monotonic method. This is because the range of dynamic priorities is usually greater than the range of static priorities, and dynamic priorities must also be recalculated at each decision point whereas static priorities never change and thus never have to be recalculated.

So, despite of the apparent dominance of the earliest deadline algorithm over the rate monotonic algorithm, the GRMT algorithms are of greater practical importance [Sha94, Lehoczky89]:

- First, they can be used to ensure that the timing requirements of the most important tasks are met when a transient overload occurs. This can be done by the technique of period transformation or by giving higher priorities to more important tasks under the constraining scope allowed by GRMT.
- Second, they provide a convenient way to offer fast response time to aperiodic

tasks while still meeting the deadlines of the periodic tasks by using the deferrable server algorithm, the sporadic server algorithm or the extended priority exchange algorithm.

- Third, they can be modified to handle task synchronization requirements by using priority ceiling protocol.
- Fourth, they can be conveniently used to schedule tasks where imprecise computation is permitted.
- Finally, they are easy to implement in (multi-)processors, in I/O controllers and in communication media.

Consequently, GRMT provides an approach to system-wide timing integration.

- **Reasoning timing correctness at an abstract level**

The advantage of GRMT can also be seen from another perspective. I. e., the major advantage of using the rate monotonic algorithm is that it allows the user to follow the software engineering principle of separation of concerns. In this case, the theory allows systems to separate concerns for logical correctness from concerns of timing correctness. In essence, the theory ensures that as long as the CPU utilization of all tasks lies below a certain bound and appropriate scheduling algorithms are used, all tasks will meet their deadlines without the programmer knowing exactly when any given task will be running. Even if a transient overload occurs, a fixed subset of critical tasks will still meet their deadlines as long as their actual CPU utilizations lie within the appropriate bound. In short, the scheduling theory allows software engineers to reason about timing correctness at the same abstract level used by, say, Ada tasking model.

## 5.2 Proposed soft real-time scheduling schemes

We are now ready to present the design of a set of soft real-time scheduling schemes which can be feasibly used in an integrated multimedia environment such as MMC.

We have designed three soft real-time scheduling schemes. They are namely a cooperative soft real-time scheduling method (CO-SCHEDULE), a semi-imperative soft real-time scheduling method (SIM-SCHEDULE) and an elastic time-scale soft real-time scheduling scheme (ET-SCHEDULE).

The CO-SCHEDULE is designed by integrating some elements from the rate-monotonic scheduling, the priority-based scheduling and the weighted round-robin scheduling. The SIM-SCHEDULE is an extension of the CO-SCHEDULE. And the ET-SCHEDULE is a novel scheduling framework into which many scheduling methods can be placed to function.

All the soft real-time scheduling algorithms presented here deal with both real-time processes and non-real-time processes.

In the following, we put our main attention to CO-SCHEDULE and SIM-SCHEDULE. A description of ET-SCHEDULE is given in the Appendix.

### 5.2.1 Cooperative scheduling by CO-SCHEDULE

A high-level presentation of the CO-SCHEDULE scheduler is shown in Figure 5-1. The CO-SCHEDULE scheduler supports a set of high-priority periodic processes and a set of normal processes. A high-priority periodic process denoted as $P^0{}_i$ has a higher priority than any normal processes and can preempt them at any time. Inside the set of the high-priority periodic processes, the processes are scheduled according to the rate-monotonic scheduling algorithm. The mapping of periods to the priorities are dynamic and are managed by the scheduling subsystem. The normal processes are again classified into different priority classes — normal priority class 1, normal priority class 2, normal priority class 3, ......, respectively, where a smaller number denotes a higher priority. Inside a normal priority class n, the processes denoted as $P^n{}_i$ are scheduled according to a weighted round-robin scheduling algorithm. That is, when scheduled to run, a process $P^n{}_i$ with weight $w^n{}_i$ is given $w^n{}_i$ time budget. Unless preempted by processes with higher priorities, the process runs until it is blocked or the time budget is used up. In the latter case, the process is placed to the end of the class n waiting queue.

The CO-SCHEDULE scheduler runs according to the cooperative time enforcement model. That is, the high-priority processes are assumed to run not longer than the worst-case-execution-time claimed by them. The scheduler schedules the processes simply according to their priorities and weight. No monitoring of the execution time of the current process is conducted.

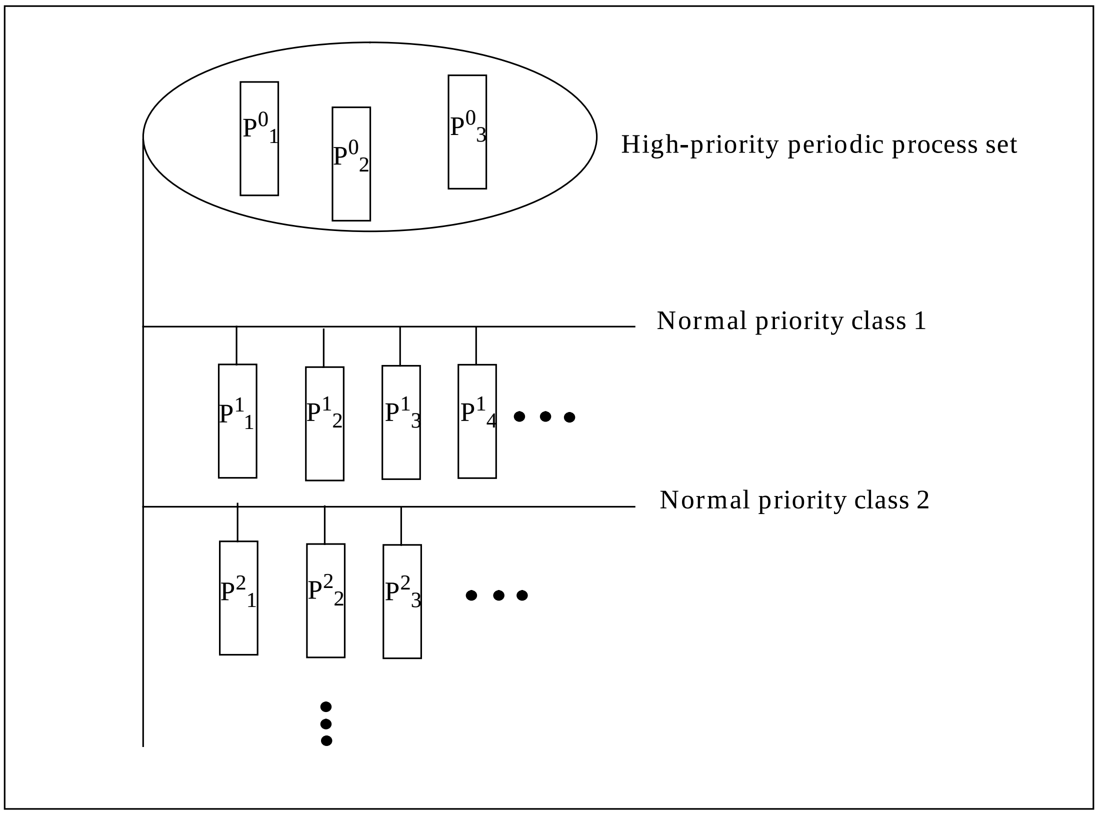

***Figure 5-1*** **CO-SCHEDULE scheduler overview**

## 5.2.2 Semi-imperative scheduling by SIM-SCHEDULE

The SIM-SCHEDULE scheduler extends the CO-SCHEDULE method by monitoring the execution of the high-priority periodic processes. The scheduler preempts a high-priority periodic process if this periodic process has not completed its task in one of its period after using up its claimed $C^0_j$ execution time. In the case of such a preemption, several following actions are then possible. For example, it can be simply allowed to complete its current execution in its next period, or it can be rendered into a process of a certain normal priority class m to continue its execution or to execute its emergency-handler, or it can be even aborted as a last resort. It is thus necessary to introduce different handling methods for timing violations to make SIM-SCHEDULE complete.

### 5.2.3 Variations and other scheduling paradigms

The above design of CO-SCHEDULE and SIM-SCHEDULE is in itself a scheme which can be further refined to produce extended algorithms.

As an extended variation, a high-priority periodic process which has used up its reserved capacity can be rendered into a normal process of normal priority class *m* before the reincarnation of its next period. This is especially useful for a virtual periodic process which should proceed as soon as possible by using *slack time* up to normal priority class *m*. Or, the different treatments of the processes which have used up their reserved capacities can be considered / combined with the handling schemes together. Still another variation is the use of a UNIX-similar multilevel feedback queue scheme to manage the normal-priority processes. Other variations may also be designed to meet the needs of specific environmental requirements.

Still other scheduling paradigms are possible. In *Section 4.2.2 Criteria of a suitable scheduling scheme*, we have pointed out the prerequisites for a scheduling method to be usable in our scheduling framework: (1) possibility to express timing constraints of processes; (2) possibility to check the schedulability of a set of processes in order to do admission control; (3) (optionally,) ability to control the processes hierachically in order to let some of the processes to be "favored" over others.

Clearly, CO-SCHEDULE and SIM-SCHEDULE fulfill all the three requirements. Many other real-time scheduling algorithms also fulfill the requirements with regard to the "mere" need of soft guarantee. In the appendix, for example, an ET-SCHEDULE based on elastic time-scale is presented.

## 5.3 Proposed soft real-time handling methods

As a soft real-time system, the timing violations from both the side of the system and from the side of the user processes are tolerated to some extent. The special characteristics of multimedia are the prerequisites for such a system to work fine. The real-time features of our proposal are soft due to mainly two aspects: first, different guarantee degrees are supported; second, the timing specifications of processes are flexible and timing exceptions are handled in some "soft" and "graceful" manners. The second aspect is necessary, since we do not require a rigid timing specification from the MM processes and not-so-often timing violations both from the side of user and from the side of the system are allowed.

By timing overflow we mean the situation where a process will apparently try to use more computation time than it previously claimed and was admitted. The timing overflow might be a consequence from the system side where the system had not given enough time to the process although the system had admitted the process with regard to the process' timing specification. The timing overflow might also be a consequence from the user side where the actual computation capacity requirement of the process would be larger than it previously estimated / claimed. Or it might be the consequences of the timing violations from both sides.

Once again, the different meanings of timing overflow for soft real-time systems and hard real-time systems should be mentioned. For hard real-time systems, timing overflows (violations) are rare and exceptional. It is **abnormal** for them to handle situations involving timing overflow. For soft real-time systems, not-quite-often timing overflows are expected phenomena. Their handling by soft real-time handling schemes is a **normal** part of the system operation.

In the case of a timing overflow, we suggest not a take an abortive measurement such as to abort the violating process. Instead, some corrective handling methods should be used to try to smooth system fluctuations.

- **Descriptions of corrective variations**

In our system, we mainly support a periodic real-time model for multimedia processes. In the case of timing overflow where the computation time in a period would apparently be larger than previously claimed by the process, different soft real-time handling methods can possibly be used:

- Method (a) — CANCEL (cancel and swallow):

  The current period entity is simply canceled by the OS. It is up to the application to “swallow it silently” by some other means;

- Method (b) — RELAY (relayed period):

  The work which can not be done in the current period is silently relayed to the next allocated period;

- Method (c) — RELAY-TRUNC (relayed period with truncation):

  As (b), but the relay can only occur maximum N times before being truncated;

- Method (d) — DELAY (delayed period):

  A prolonged period is allowed but at the cost of a delayed new instantiation of the period following this one.

Method (d) has the advantage that the process in question can get what it immediately needs. But this method violates the guarantee constraints allowed by GRMT, since the prolonged period implies a larger usage requirement. That is, the usage bound for a certain scenario might be exceeded and this might lead to the result that other processes are negatively influenced. Therefore, the “delayed period” method is an opportunistic method which should usually not be applied.

Methods (a) (b) (c), in contrast, try to maintain the system in such a way that the violating process will not have a negative influence on other processes.

Methods (a), (b) and (c) can be augmented by a timing-violation handler (TV-handler) which is triggered when a timing-violation occurs. By triggering such an exception handler, the scheduling subsystem notify an application process that it is unlikely to schedule its activity according to its time-constraints. This mechanism allows the application processes to react to the missed deadlines in an application-specific manner. Such a notification mechanism is one of the key features with which an adaptive service supporting model can be built. Such a usage is further refined in *Section 9.3.4 Adaptation support from OS scheduler*.

These handling methods can be used in direct combination with SIM-SCHEDULE since a timing violation monitoring is conducted by the system. Their combinations with priority manipulations in the scope allowed by GRMT are also possible. The ideas of timing violation handling are also partly applicable to a system controlled by CO-SCHEDULE if a self-monitoring is done by the processes themselves.

In order to evaluate the soft real-time framework and the related soft real-time scheduling and handling methods, both simulations and implementations have been conducted. The simulation construction and the related results will be detailed in *Chapter 6* “*Simulations of soft real-time effects*”. The experimental implementations of these schemes and the related measurements and assessments will be the topic of *Chapter 7*.

## 5.4 Usage issues

The following discusses the usage schemes for the above soft real-time scheduling and handling proposals. They also reflect some further justifications for our design.

### 5.4.1 Implications for the implementation schemes of applications

One of the most important design decisions in the above soft real-time scheduling schemes is the explicit and direct support for real-time periodic or virtual real-time periodic processes. This influences the implementation schemes of the applications directly.

- **Periodic framework**

Periodicity is a common property of real-time tasks in continuous media applications. The decoding and playing-back of a series of video frames coded in motion-JPEG is such an example. Since periodicity is a typical feature of the MM activities, the proposed scheduling methods provide direct “sure guarantee” and “maybe guarantee” aupport to periodic or virtual periodic processes.

A main concern related to a periodic processing framework for continuous media is that the arrival patterns, data volumes and processing capacity requirements may vary from period to period. But as pointed out by some related work such as [Steinmetz95, Jeffay95], such variations do not usually lead to the infeasibility of a periodic framework. Instead, some forms of periodicity are always exploitable.

Although the periodic processing time for multimedia applications does vary, it is usually still controllable in the following senses:

a) It does not vary very much in many cases. For example, the statistics on the processing of a series video frames coded in JPEG or a series of video frames consisting of I-frames usually indicate a relative stable curve with small variations.

b) The stable periodicity may appear in a somewhat transformed form. For example, the processing time for a series of MPEG frames consisting of I-, P-, and B-frames may vary greatly. But from the view point of GoPs (group of pictures, which consists of all the frames related to an I-frame), the periodic processing time is usually relatively stable. Figure 6-4 on page 137 will show such an example.

c) The related communication-patterns concerning data volume and arrival pattern are usually also of periodic manner. See, for example, [Vogt95].

In view of these observations, small variations can be easily contained and tolerated in the elastic framework of soft real-time. Major mode changes can be dealt with by the adaptive service model of Chapter 9.

- **Applying periodic model to implementations**

Naturally periodic applications can be implemented in the periodic scheme easily. Non-periodic applications which need guarantee can be rendered as pseudo-periodic processes to fit into the framework.

Two most important indices of a periodic process are its period T and its per period process time requirement C. The ratio C/T indicates the processing bandwidth the periodic process requires. The processing bandwidth is critical for the admission control under GRMT. One of the biggest advantage in using a periodic model to realize the application is that the timings are not bound to single events. Instead, a natural periodic timing framework is provided to contain the needed timing events. This feature is especially suitable for the easy realization of the soft real-time applications such as A/V applications.

Above all, the A/V applications posses some natural periods which are directly mappable to process periods. These periods can be estimated independently of the concrete computer and OS platforms. For example, a decoding process for video frames must process 30 frames per second. A natural period for it would be 33.3 ms. An audio process may have to be scheduled every 50 ms to generate an audio buffer. Sometimes, a period transformation may have to be applied. For example, the audio process may have to be submitted as a process with a period of 100 ms (for 2 buffers) in order to lower its relative priority to other processes with shorter periods.

The processing time needed for each period is unavoidably platform dependent. Some methods may be used to make an estimation of per period processing time easier. In our prototype system implementation, for example, the processing time can be estimated gradually in a “try and asses” cycle due the property of our scheduling subsystem as a “timing enforcer and predictor” in one (Section 7.3.3). Such a method can be very helpful for supporting the user to adapt their applications to different platforms with different hardware and OS configurations.

- **Run a non-periodic process in the periodic framework**

Not only the naturally periodic processes can be run in the periodic scheduling framework. There is also the possibility of providing virtual periodic processing service.

Actually, many system management and application functions should be ensured to progress deterministically. Though they may not be directly MM-relevant, they still need virtual real-time support. I. e. they should also be guaranteed with a certain share of processing bandwidth. For example, in a MMC scenario, we may want to guarantee the file transfer and processing to accompany an interactive video connection.

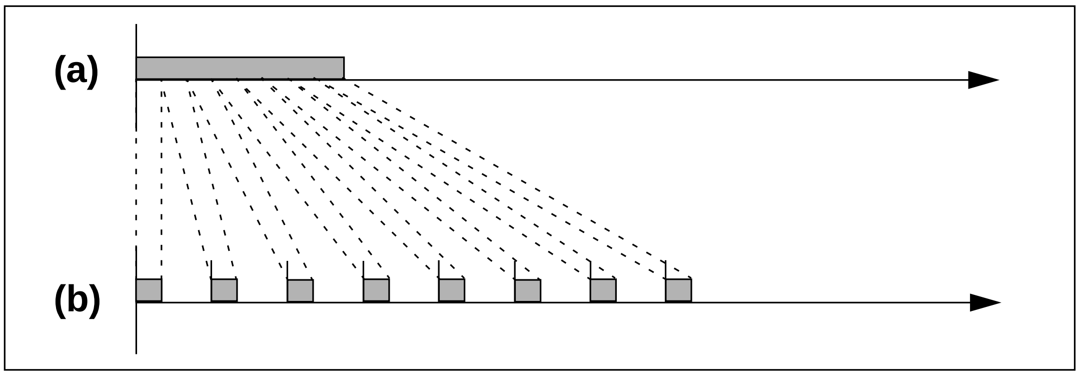

***Figure 5-2*** **Fitting non-period into the periodic framework**

Note that, in order to check schedulability bound in a scheduling system using rate-monotonic scheduling, a non-periodic process has also to be considered and scheduled in the same way as a periodic process. In order to run a non-periodic process in our periodic scheduling framework, the scheduler has to schedule the non-periodic process in an "artificially periodic" way. Since non-periodic programs do not have an explicit periodic structure and requirement, they can be handled by arbitrarily setting the period computation time and period to yield an intended rate, as depicted in Figure 5-2. For example, a compilation job has to be guaranteed of 3% of processing bandwidth. It may be rendered into a process of 3ms processing time in every 100ms or a process of 12ms processing time in every 400ms.

In this way, such important non-periodic processing can be assigned the role of virtual periodic process with sure- or maybe-guarantee, dependent on their importance.

The concrete forms of usage of periodic processes and virtual periodic processes are exemplified by our experimental implementation and are detailed in *Section 7.2.2 Sample programming interface of usage*.

### 5.4.2 Implementation methods of periodic bodies

Within the periodic framework, the users of the periodic process model have two choices in realizing a periodic entity.

The first possibility is to use a periodic process with the timing properties managed and enforced by the scheduling subsystem. Using this method, the user provides the periodic body which conducts the work expected in a period. For example, in any single period of 25ms with processing capacity up to 4ms, the periodic body will normally process 4 messages and will process up to 6 messages if more messages are available. The application should provide the scheduling system with the period and per period usage it perceives. The whole process can then be created and managed by the scheduling subsystem automatically. The reinstantiation of the periodic body and the timing-overflow handling will be dealt with within the scheduling subsystem.

The second possibility is to use a virtual periodic process and to manage the timing properties by oneself. The application should still provide the scheduling system with the per period usage it requires. The scheduling system will only enforces the per period maximum usage of the virtual periodic process. The timing-overflow handling will then not be dealt with within the scheduling system, since the boundaries between the (instantiations of the) periodic bodies are intentionally left out. In the periodic body of this case, the application should use the basic timing facilities provided by the system to decide whether and when to activate their own timing-overflow handling.

### 5.4.3 More on scheduling of normal processes

Periodicity is a common property of real-time tasks in continuous media applications. The high-priority periodic process set supported by CO-SCHEDULE or SIM-SCHEDULE can, for example, be used to implement functions for isochronous data transmission and consumption. The set of the high-priority periodic processes can be guaranteed to meet their timing specification if the processor usages of the set are checked to be in the schedulability limit, as given, for example, by Theorem 4-1, Theorem 5-2 and Theorem 5-3.

Note the processes in the normal process class can also be expected to run in some sense of real-time if carefully planned. For example, process $P^1_i$ will be granted of its required $w^1_i$ computation time in a cycle of L, if the following equation holds:

$$\left(\sum_{j=1}^{n_0} C_j \left\lceil \frac{L}{T_j} \right\rceil\right) + \left(\sum_{k=1}^{n_1} w^1_k\right) < L \qquad \text{(E-5-3)}$$

where $n_0$ and $n_1$ are the number of the high-priority periodic processes and of the priority class 1 processes respectively. $T_j$ and $C_j$ are the period and per-period computation time for $P^0_j$. $w^1_k$ is the weight of $P^1_k$.

As explained in *Section 4.1.2.3 Timing enforcement models* and *Section 4.1.2.4 Enforcement on periodic processes*, the predicted schedulability of the proposed schedulers can only be achieved if timing monitoring and enforcement measures are taken. Only in this way can the intentional or non-intentional mal-function of one process be prevented from having negative effects on other processes.

### 5.4.4 Jitter of processing

CO-SCHEDULE and SIM-SCHEDULE both assume a periodic process model for CM-related processing and schedule the high-priority periodic processes according to the Generalized Rate Monotonic Theory.

In a complicated scheduling scenario, processing jitter is sometimes inevitable to some of the periodic processes. This can be seen in the following figure.

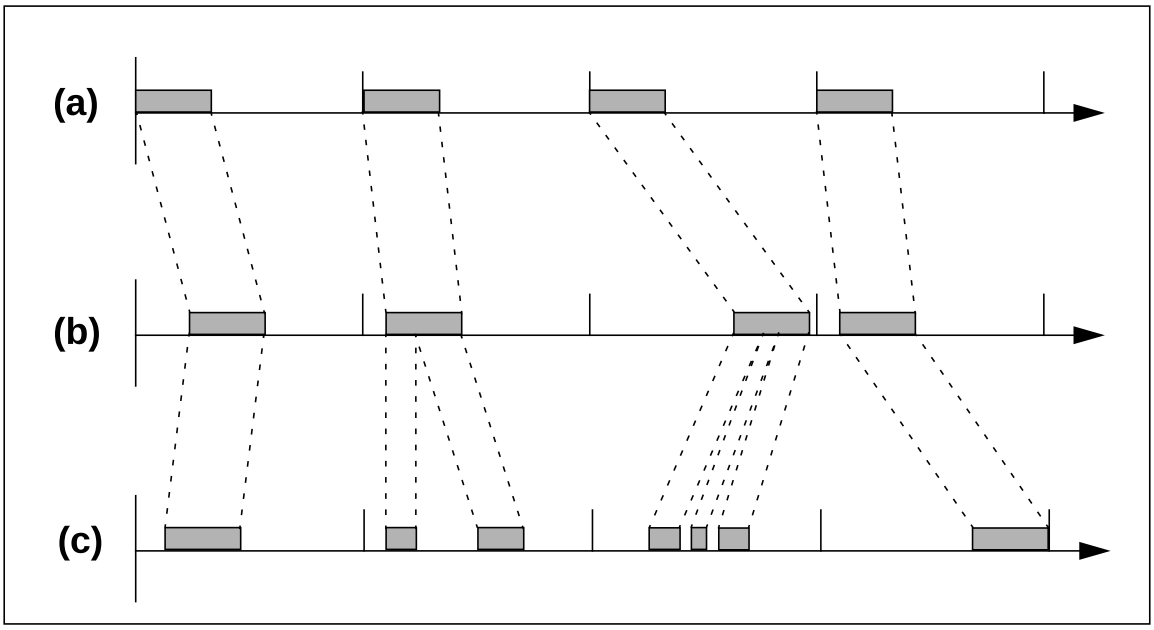

***Figure 5-3*** **Actual processing usage in a scheduling scenario by RM**

Rate monotonic scheduling guarantees that a periodic process gets its reserved per period processing time in its cycle of periods. But the processing time the process receives may not always at the same phase of its period (say, at the beginning of each period). It is also possible that the process gets its processing time not in a chunk, but in several pieces. The reasons for these phenomena are that the scheduling is done according to rate-priority and that the context switch is unavoidable due to the interweaving of different periods and phases of different processes.

One may try to alleviate the problem by trying to keep the processes ready for scheduling. For example, a delayed playback in a communication and processing application with periodic message arrival can be sure that it always has some messages to process. A MPEG-decoder retrieving its data from disk may starts its I/O request in the i-th period and begins its processing of the read-in data in (i+k)-th period when the requested data are sure to be ready. But the processing jitter is still unavoidable for the processes with lower rate-priorities whose processing can possibly be preempted by the processes with higher rate-priorities.

It should be pointed out that the jitter of processing is also unavoidable if the processing is scheduled according to a scheduling method like earliest deadline first (EDF). To schedule a set of periodic processes according to EDF, the scheduler will just try to give the period instance with the earliest deadline to run. Again, due to the interweaving of different periods and phases of different periodic processes, the process can neither be guaranteed to receive its processing time always at the same phase nor in a chunk or in several pieces.

For the processes which are used at the intermediate stages of the MM processing, such jitters of processing are usually not a big concern. The intermediate stages must only be sure that they are making regular progresses and that the related jitters of processing are under control (actually, always in the scope of maximum one period).

Jitter of processing is, however, a big concern for the end-processes which have to interface audiovisual devices more or less directly. For them, jitter control at the process level is sometimes necessary to match related physical requirements of hardware devices. Some of the jitter-control techniques in the scope of GRMT [Klein93] can be used for this purpose. For example, a process with short period (thus high rate-priority) may be used to conduct I/O processing only. Or, a process can be assigned a temporary higher priority for an intended short section of I/O-related processing.

In “*Section 5.5.3.2 Interfacing continuous media I/O*”, further issues concerning interfacing logical and physical media devices will be discussed.

## 5.5 Realization issues

One of the overall goal in implementing our soft real-time framework is to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system can be run in a more or less predictable manner. By exploiting the soft real-time properties of the CM applications, this approximation can usually be realized in simple and efficient ways. In addition, the management of other resources and activities should be taken into account so that the real-timeliness and predictability of the system as a whole can be achieved.

### 5.5.1 Implementation schemes with different OS structures

Above all, the first main realization issue is the efficient implementation and provision of the base real-time scheduling services and the diverse soft guarantee semantics. As pointed out by [Stankovic92], it is principally possible to achieve hard real-timeliness with different OS structures. Likewise, it is principally possible to realize soft real-time by approximating the realization of hard real-time mechanisms on different OS architectures.

It should be noted that the term “real-time OS” used here is only a general concept. It practice, this “real-time OS” may be implemented as a real-time OS of its own, it may be implemented in some environments as an extended run-time system based on the extension of an available OS or it may be simply implemented as a set of supporting library modules (especially for an embedded environment).

In the following, different advantages or difficulties in approximating soft real-time on different OS architectures are considered. Comparisons are made with regard to monolithic kernel, micro-kernel and nano-kernel OS structures (see Figure 5-4). Our preference for a micro-monolithic-kernel is then justified and detailed.

#### 5.5.1.1 Monolithic kernel vs. microkernel

As long as the structure of a real-time OS for multimedia is concerned, a first comparison between monolithic structured and micro-kernel structured OS should be conducted.

In a microkernel-based operating system structure [Gien91, Hildebrand93], the micro-kernel provides system servers with generic functions. The functionality of a micro-kernel is limited to generic processor scheduling and memory management functions, usually independent of a particular operating system environment, and a simple inter-process communication (IPC) facility that allows system servers to interact independently of where they are executed, in a multiprocessor, multicomputer, or network configuration. System servers, in turn, are autonomous OS service function providers on top of the micro-kernel. A modular set of servers provides OS services and OS interfaces to application programs. These services, such as file system, devices and high-level communications, are traditionally incorporated in the kernel of a monolithic system. This combination of primitive services from a standard base, which in turn supports the implementation of functions that are specific to a particular operating system environment, makes a micro-kernel based OS structurally modular.

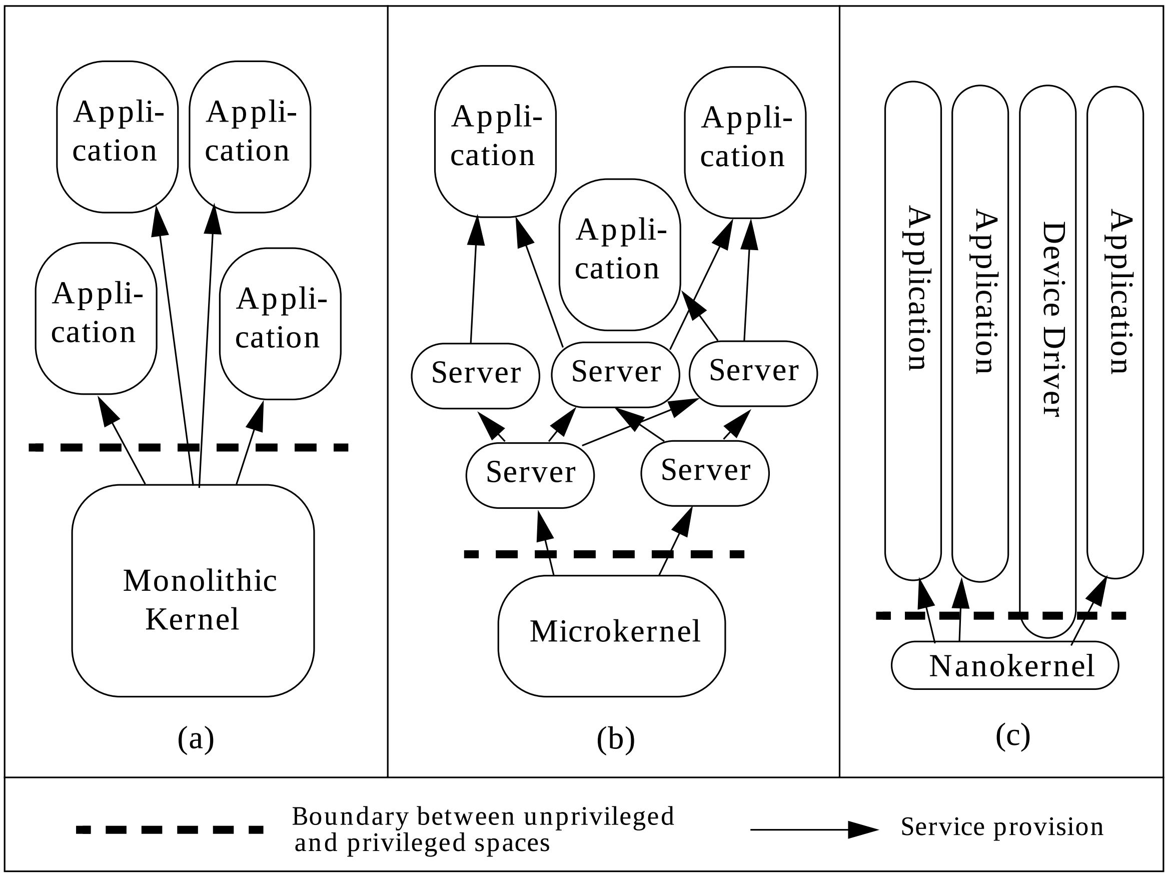

***Figure 5-4*** **Schematic comparison of three different OS structures**

In contrast, traditional monolithic-structured OS is generally inflexible for real-time applications. The main problem lies in the monolithic kernel where most of the system services are provided through system calls which are traditionally single threaded and non-preemptable. Longer systems calls can thus be a big hindernis for real-time response.

For traditional monolithic OS, real time responsive ability can usually be introduced by using preemption point mechanisms everywhere in the kernel implementation, e.g. in the system calls. But such added-on real-time capabilities are usually quite limited in both functionality and performance. Compared to monolithic OS, a micro-kernel based OS is structurally preemptive because of its modularity, i.e., it has a structural advantage in that the micro-kernel OS can reduce unexpected delays in the kernel when other tasks execute expensive system calls.

Thus, from the viewpoint of the traditional industrial real-time system where a short responsive time is a key factor, it seems that micro-kernel OS is more suitable to serve as a base for real-time systems.

In the context of soft real-time multimedia support, however, the above advantages of micro-kernel structures play much less significant roles. Instead of them, the emphases should now be put on direct MM-support service interfaces, fewer multiplexing points to minimize QoS crosstalks between applications and easy reservations with simple accounting. (By QoS crosstalk, we mean the phenomenon where the QoS of an application is negatively influenced by other concurrent applications, especially in the cases of overloads.) Actually, as far as these new aspects are concerned, the micro-kernel structure has more disadvantages than the micro-monolithic structure explained below.

In the micro-kernel structure, much functionality is moved from the kernel into different server processes. These servers are then multiplexing service provision points. It is then more troublesome to do resource reservation and usage accounting across these different server domains than centrally by a (micro-)monolithic kernel.

#### 5.5.1.2 Vertically integrated approach with nano-kernel

One of the trends in modern operating system technology is to exploit a more application-oriented architecture. The foremost reason for it might be that it is then easier to optimize the system in a more application-specific way. An extreme example of such a trend in the area of multimedia support is the vertically-integrated nano-kernel structure as advocated in [Roscoe95, Leslie96].

In this radically different approach, each application is virtually a self-contained protection and execution domain. Instead of system services being located in the kernel or server processes, they are placed as much as possible in client application protection domains and scheduled as part of the client. Communication between domains occurs only when necessary to enforce protection and concurrency control. This amounts to multiplexing the service at as low a level of abstraction as possible.

Such a structure raises a number of technical issues. Programmers should neither have to cope with writing almost a complete operating system in every application nor contend with the minimum level interfaces to share servers. Binaries should not become huge as a result of the extra functionality they support and resources should be allocated in such a way that applications can manage them effectively.

In order for this approach to work efficiently, the above technical problems should be solved. First of all, this architecture should be generally coupled with the single-address-space scheme in order to achieve a high degree of data sharing thus to make the system efficient enough. Structuring tools in the form of typed interfaces within a single address space should, for example, be used to reduce the complexity of the system from the programmer's viewpoint and enrich sharing of text and data between applications. Second, processing bandwidth should be scheduled between the domains in a guaranteed manner, the domains should be aware of their CPU allocation and the CPU multiplexing within application domains should be easy. Third, inter-domain communication methods should be carefully designed to allow efficient communication without complicated binding and invocation.

### 5.5.2 A micro-monolithic-kernel approach

While agreeing with the lines of thoughts of the above vertically-structured nano-kernel structure, we hold the opinion that it is generally not necessary to go that far as to have to deal with those many new problems of the new structure. As discussed in the previous chapters, the soft or elastic features of continuous media can, after all, tolerate QoS crosstalks to quite some extent. We advocate, therefore, a more traditional micro-monolithic-kernel structure where many traditional system call kernel functions are implemented as user-space libraries so that the monolithic kernel can be kept quite small. (It is possible to perform most of the kernel functions of an operating system like UNIX in user-space applications. Example can be found in the work like the Spring SunOS emulator [Khalidi92], where the whole SunOS is almost entirely implemented as a client library.)

Note that many existing real-time systems tend to be built along similar lines as our micro-monolithic approach, with a very simple generic executive supporting application-specific real-time tasks.

#### 5.5.2.1 Advantages of the approach

Using the micro-monolithic approach, the system structure is simple. This leads to simple reservation and accounting. The advantage of micro-kernel can also be achieved to some extent.

First of all, rescheduling/context switch can be done without the influences of some long-delaying system call executions. Second, the cost of invoking library functions is accounted directly as part of the capacity reserved by the user processes. (It is true that the costs of these library functions should be measured and their behavior under different conditions should be verified. But such analysis should usually be done only once. The statistics can then be stored and provided for all the future use). Third, the capacity used by the system kernel on behalf of a user process should be “paid” by the user process and it is far easier to do such accounting in a monolithic way than in a micro-kernel environment encompassing several (sometimes recursive) server domains. Thus, the accounting complication by a pure micro-kernel structure [Mercer94a] can be avoided to a large extent. Fourth, the system should usually provide “advisory” cost information with regard to the execution of MM-related library functions and system calls in order to aid the user processes in reserving their processing capacities. The reason is that it is sometimes quite difficult for the processes to estimate their costs for all circumstances accurately. This difficulty can be alleviated to some extent if the scheduling subsystem can be designed to be time-constraint predictor and time-constraint enforcer in one. With the aid of such a scheduling subsystem, the user processes can undergo a "try and revise" procedure to attain a good estimation of their actual processing capacities. With the simplicity of accounting in a monolithic system, it is usually easy to achieve such a goal, as to be exemplified in *Section 7.3.3 Scheduler as a predictor and an enforcer*.

Additionally, some of the structural advantages of the micro-kernel structure can be emulated in the micro-monolithic structure to some extent by way of kernel multithreading. In our opinion, micro-kernel and real-time are somewhat orthogonal in functionality. The key point is multithreading as long as the organization of the system activities is the main concern. Traditional "system" operations are not necessarily more important than other "user" activities. They should be threaded to compete for CPU and other resources with other user activities to achieve the real-timeliness of the whole system. Such multithreaded processing model can be implemented in the form of a micro-kernel structure or a multithreaded kernel, as is shown in Figure 5-5. If a pure micro-kernel approach is pursued, great efforts should be taken to implement real-time servers [Nakajima93]. Such efforts can be spared in the micro-monolithic structure.

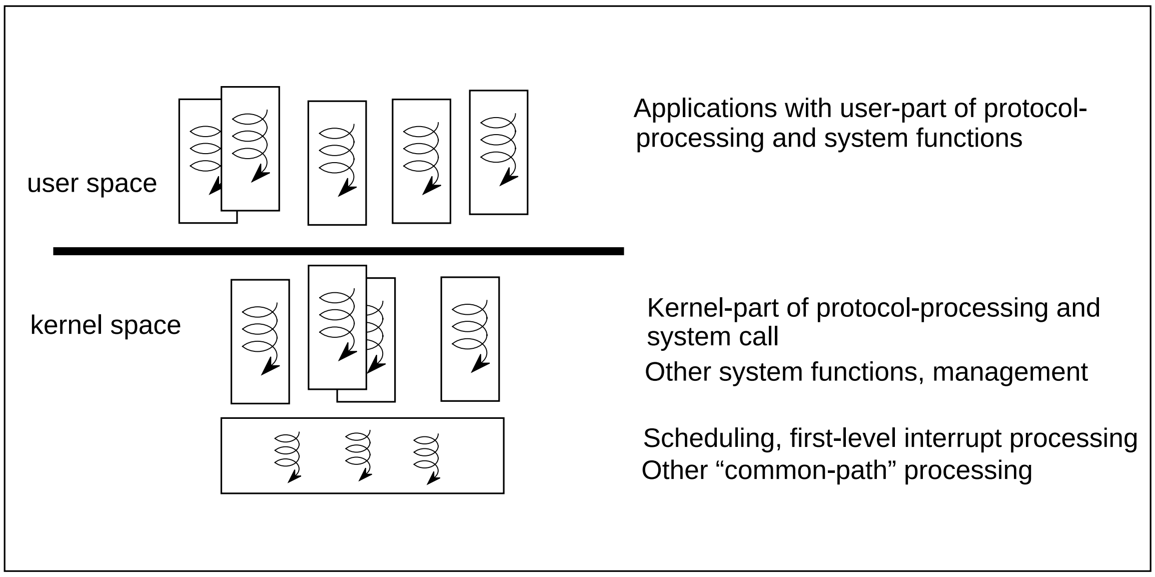
user space
Applications with user-part of protocol-processing and system functions
kernel space
Kernel-part of protocol-processing and system call
Other system functions, management
Scheduling, first-level interrupt processing
Other "common-path" processing

***Figure 5-5*** **System overview with a micro-monolithic- and multi-threaded-kernel**

In our experimental implementation, for example, some system management operations are embedded in a system process of normal priority. Their executions will, therefore, not influence the executions of (virtual) periodic MM processes. This will be exemplified in *Section 7.3.6 Other features of interest*.

#### 5.5.2.2 Considerations on the processing cost for common paths

Similar to the vertically integrated nano-kernel approach, in a micro-monolithic system, the activities belonging to an application should be reserved with regard to the application and should be accounted to the application as much as possible. This means that the owner of the activities should be identified as early as possible. Common paths should, in contrast, be minimized.

In a micro-monolithic system, it is quite straightforward to reserve and account for the processing capacity used to execute plain application codes, library functions and system calls invoked by the application. It is not so apparent as to how to deal with those common path executions which are not identifiable with a certain application or an independently reserved system function. The initial phase of an interrupt processing and the scheduling overhead of the scheduler itself are two such common path executions.

Generally, in the cases where the amount of cost for such common path executions is quite small, it might be taken as part of the system variations which should be tolerated by the soft-guarantee semantics of the soft real-time framework. In the cases where the amount of such cost is too significant to be ignored, however, the capacity for such cost should be independently reserved and accounted.

In a scheduling subsystem controlled by CO-SCHEDULE or SIM-SCHEDULE, the reservation of a fraction of processing power for a small common portion of system activities can be made relatively easily. Again, in the case where the amount of such cost is quite small, it can be taken as part of the system variations which should be tolerated by the soft-guarantee semantics of the framework. In the case where the amount of such cost is too significant to be ignored, the capacity for such cost can be reserved by reserving the processing capacities for one or more virtual “system-overhead” periodic processes with very short virtual periods (virtually shorter than the periods of any periodic processes). According to the rate-monotonic theory, these virtual system-overhead processes can break the executions of other processes at any time when they are ready, because they have the highest priorities resulting from their very short periods. This has the practical effect that the virtually arbitrary breaks caused by the scheduling and interrupt handling are, to a large extent, compensated by these virtual reservations. For a simple admission control based on GRMT, this has the simple form of just reducing the schedulability bound by the amount of the reserved system-overhead capacity.

With communication-intensive applications, the cost of interrupt processing caused by incoming packets can be quite significant if the protocol processing is implemented in the traditional kernel/server way. It is the topic of Chapter 8 to minimize the common path processing involved in the protocol processing and to make the protocol processing more controllable and predictable.

If not otherwise indicated, all our following discussions will be made with the micro-monolithic approach as a reference. Especially, the experimental implementations to be documented in Chapter 7 also follow this approach.

### 5.5.3 Real-timeliness and predictability of the system as a whole

A multimedia application is dependent on several kinds of resources in its whole life time. Processing capacity is only one of the most important active resources. Resource management and allocation should, of course, deal with all resources in a multimedia system. Real-timeliness and predictability of the system as a whole can only be achieved if all system resources and activities involved are managed in a predictable manner. The issues of interfacing audiovisual I/O and making protocol processing more controllable are also directly relevant.

#### 5.5.3.1 Application- and timing-driven resource reservation

It has been argued that it is natural to do resource reservation in an application-driven approach, since the QoS requirements of the applications dictate the resources required by underlying system components which service this application. In addition to this, resource allocation should also be done in a timing-driven manner. Since real-timeliness is a key factor in continuous media support, the resource allocation and usage should be surrounded around timing consideration.

In the scope of an endsystem, CPU is a kind and actually the most important kind of resource as long as timing is concerned. In order to achieve the real-timeliness of the whole system, all the system activities should be controlled by the real-time management scheme. We have proposed to give CM-related processes some kinds of soft guarantee of their CPU usage. Admission control with related real-time scheduling is the “reservation mechanism” which we have used to offer the soft guarantee. For a capacity-based scheduling system such as GRMT scheduling, the relationship between CPU capacity reservation and guarantee is quite apparent.

Since the processes also use other kinds of resources, these resources are directly or indirectly involved to make the system as a whole predictable. Soft real-time scheduling and handling is only the reservation of CPU capacity resource. The allocation (sharing or reservation) of other resources should also be done by taking timing factor into consideration. Only in this way can the system behave predictably as a whole.

Some resources such as bandwidth to a peer endsystem will involve peer and intermediate systems. The management of such resources will inevitably have to be conducted in a distributed manner. The reservation and enforcement of the transmission bandwidth in an integrated service network, for example, can be conducted by use of RSVP, CBQ, etc. [Fan97].

The management of the local resources on an endsystem should be done by taking timing requirements into consideration. For example, to reserve some passive resources such as buffer memory, methods used in [Ferrari92] should be applied along the time axis of the applications’ life time.

#### 5.5.3.2 Interfacing continuous media I/O

Chains of processes may be involved in a multimedia application. Usually, the end-processes will have to interface continuous media devices more or less directly. For them, it is usually necessary to match their processing speed and processing rate to the related physical requirements of some logical or hardware I/O devices. This stands in contrast to the intermediate stage processes which usually have only to be sure that they are making regular progresses and that the related jitters of processing are under control.

The problems with the input direction from and the output direction to a CM device, typically an audiovisual device, are usually different. Interfacing input from a CM device is generally not very problematic where the process is generally the single user of the device. The hardware of the audiovisual device generates the input data periodically and the generated data can generally be delivered to the process through a notification mechanism such as interrupt or status bit. Many CM input devices support a FIFO buffer to further simplify the timing requirement of the input process. On output, in contrast, the case is generally more complicated. The output CM device is usually a place where some mixing and multiplexing should be done. For example, several audio streams may have to be mixed before being playing out by a loud-speaker and a window system usually has to support multiple video windows.

- **Audio input and output**

Modern workstations and high-end PCs usually have built-in audio support. On a Sun SPARC, for example, the 16-bit audio device can be used in almost the same way as a standard I/O device. The *`/dev/audio`* device can be *`open`*ed and then asked to *`read`* and *`write`*. Parameters such as sampling rate, gain, channels can be set and retrieved by *`ioctl`*. The *`/dev/audioctl`* device can be used to set volume. For audio input, an event-based technique such as using *`select()`* can be used to wait for several possible events such as the readable state of the audio file descriptor.

For audio output, it is feasible to use an explicit periodic timer to trigger the writing of blocks of audio data to the audio output (such as a speaker) or use a periodic process to do the job. Since there is usually a FIFO buffer to the speaker (such as on a Sun workstation), the preciseness of the period of the writer to the buffer can then be relaxed a little — the writer will then be regulated (blocked/unblocked) automatically by the high-water and low-water mark mechanism of the FIFO. In the NEVOT Internet audio terminal [Schulzrinne95b], a more efficient method has been used to do the timing for audio output. It does this by coupling the audio input and output timing: each time the audio input device (the analog-to-digital converter) delivers a full block of audio (usually, corresponding to 20 ms) to the input application, one equal duration block of audio data is copied from the playback buffer to the audio output device. This method has the advantage that it is not necessary to rely on the system clock for the accurate periodic timing and it also incurs less overhead.

By the way, a play-back process used to handle audio play-back usually adjusts its play-back delay quite often — sometimes even with each new talkspurt. A scheduling model based on GMRT (such as ours) guarantees the processing bandwidth of the process but does not relate a user-relevant timing decision (in this case the play-back delay) with the scheduling. It brings with it the advantage that such user-relevant timing decisions will not influence the system scheduling directly or frequently which may be the case with a scheduling scheme based on EDF etc.

- **Video output and real-time window system**

Video and animation outputs are usually sent to a window system to be displayed. There are problems with a traditional window system like the X window system. For some decoding and displaying experiments conducted on a Sun SparcStation, for example, the main processing power has been found to be devoted to the video decoding itself. But a significant part (varying from 5% - 30%, dependent on the display content and the number of the applications) of the processing power is consumed by the X-server which displays the decoded results. This part of processing power is not accounted to the video applications. In addition, the X-server handles the client requests basically in a FCFS manner.

Several implications follow. First, if several video streams have to be displayed concurrently, then the X-server does not take into account their QoS requirements. A stream may overwhelmed the others. Second, the display requests from other non-real-time applications may delay the intended normal delay rates of the video streams. In an overloaded case, both situations become worse.

There are basically two approaches to solve the above problems. One approach is to have the applications take over most of the rendering work of a window system, reserve and account such work to the application itself. The work concerning other window management functions can be left as it is and be viewed as non-real-time. The other approach is the so-called real-time window system approach where the window system is multithreaded and correspondingly scheduled in a real-time manner.

A recent version of the $8^1/_2$ window system of Plan9 renders graphics almost entirely within the client. The client then sends bitmap tiles to a window manager which is optimized for clipping these tiles and copying them into the frame store. And, in some experimental implementations like the Cambridge Desk Area Network, many low level window manager primitives have been provided in hardware. If such techniques are combined and applied together with the micro-monolithic kernel, then the reservation and accounting of video output are quite application-driven and can be easily done.

The feasibility of the real-time window system can be seen in some experiments like [Sasinowski95]. Some real-time threads and non-real-time threads are created as the processing entities for the window server. Real-time threads only process the categorized real-time primitives such as drawing operations and event-related requests, which are usually of deterministic length and which are directly real-time relevant. Real-time window threads and other real-time activities are scheduled by some real-time scheduling schemes. Although the relationship between the RT window threads and their clients can be diverse in a general real-time window system, it will achieve the best effect if a client application needing real-time window service can provide its window-related QoS requirements before hand and the client processes together with the corresponding RT window thread can undergo the admission control together to make sure that the system has the capacity to support them satisfactorily.

#### 5.5.3.3 Protocol processing and mode changes

In addition to the application MM processes, the protocol processing activity is one of the main activities on a MM communication system and it should also be managed by the system in a controlled manner. Especially, the interrupt processing cost caused by communications is usually quite significant. Several degrees of treatment are possible: (a) consider interrupt-driven protocol processing as system noises; (b) try to damp the interrupt effects; (c) use special hardware such as front-end to distribute part-load; and (d) exploit application-driven protocol processing. These are different methods of bringing the protocol processing under control.

Another major issue in making the whole system predictable concerns mode changes requested by the application itself or caused by variable environmental conditions. It is possible to support a flexible and adaptive service model (the FAST model) in our soft real-time framework. The FAST model can support both the application-initiated adaptations and the supporting-system-initiated adaptations needed by multimedia communications and applications.

Because of the importance and the scope of these two issues, Part IV will have a detailed treatment of them to make our soft real-time framework complete.

## 5.6 Summary of the chapter

The center of a soft real-time framework is a set of feasible soft real-time scheduling and handling schemes. The basic requirement on these schemes is that they should be simple, predictable, flexible yet powerful.

The Generalized Rate Monotonic Theory (GRMT) has been chosen as the theoretical basis for the design of our soft real-time scheduling and handling schemes. The choice of GRMT is based on mainly two reasons: the completeness of the GRMT theory for many real-time environments; and the practical advantages of the GRMT-oriented real-time schemes over many other real-time methods.

Our scheduling schemes are designed by integrating some elements from the rate-monotonic scheduling, the priority-based scheduling and the weighted round-robin scheduling. The Generalized Rate Monotonic Theory is used in all the methods as a basis for admission control and for real-time scheduling. The other related results of the theory are also used in other aspects of the system implementation such as preventing unbounded priority inversion by some forms of priority inheritance.

Two soft real-time scheduling schemes are then presented which can be feasibly used in an integrated multimedia environment such as MMC. They are namely a cooperative soft real-time scheduling scheme (CO-SCHEDULE) and a semi-imperative soft real-time scheduling scheme (SIM-SCHEDULE). Other than their different timing enforcement models, the basic design of CO-SCHEDULE and SIM-SCHEDULE are similar. Their scheduler supports a set of high-priority periodic processes and a set of normal processes. Inside the set of the high-priority periodic processes, the processes are scheduled according to the rate-monotonic scheduling algorithm. The normal processes are categorized in classes and are scheduled in the weighted round-robin algorithm.

For soft real-time systems, not-quite-often timing overflows are expected phenomena. Their handling by soft real-time handling schemes is a normal part of the system operation. A set of corrective soft real-time handling schemes has been proposed which will try to smooth system fluctuations: CANCEL, RELAY, RELAY-TRUNC and DELAY. These handling schemes can be used in direct combination with SIM-SCHEDULE and are partly applicable to CO-SCHEDULE.

The above soft real-time scheduling schemes provide an explicit and direct support for real-time periodic or virtual real-time periodic processes. This influences the implementation schemes of the applications directly. Usage issues have been discussed concerning applying the periodic model to implementations, running a non-periodic process in the periodic framework, implementing periodic bodies and handling jitter of processing.

One of the overall goal in implementing our soft real-time framework is to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system can be run in a more or less predictable manner. In order to achieve an efficient implementation and provision of the base real-time scheduling services and the diverse soft guarantee semantics, we advocate a micro-monolithic-kernel structure for its advantages over traditional monolithic, micro-kernel or vertically integrated nano-kernel approaches.

Real-timeliness and predictability of the system as a whole can only be achieved if all system resources and activities involved are managed in a predictable manner. Some issues are explored concerning how to conduct application- and timing-driven resource reservation and how to interface continuous media I/O. The issues of making protocol processing more controllable and dealing with mode changes by adaptive service provision are delayed to the next part for a detailed treatment.

In short, the chapter has shown how to concretize a soft real-time framework by exploring the issues in designing, realizing and using a set of feasible soft real-time scheduling and handling schemes. It is the foundation for the further work on qualitative simulations and experimental implementations of the next chapters.

# Chapter 6

# Simulations of soft real-time effects

The main purpose of our work on simulation (described in this chapter) and implementation (described in the next chapter) is to further ascertain the feasibility of the soft real-time framework and to evaluate the related soft real-time scheduling and handling methods — not only qualitatively but also quantitatively, not only theoretically but also practically.

## 6.1 Simulation with practical implications

The construction and realization of our simulation models as well as their parameterizations are based on extensive analyses of the practical application scenarios and current multimedia software packages. Some of the analyses have been mentioned in *Section 2.1.4 Implementation schemes of typical applications* and *Section 3.2.1 Modeling CM activity as play-back application*. In addition to the components reflecting soft real-time scheduling and handling methods, our simulator has reflected some of the characteristics of continuous media applications such as buffer sizes and playback delays. It has also reflected some of the characteristics of the networking environments such as delays and losses.

For example, some observations on the MBONE audiovisual data transmission over the Internet have shown the following facts: many packets are simply missing, many packets are misordered, but usually no packets are duplicated. The histogram in Figure 6-1 shows a section of the packet arrival situations across the Internet. It was recorded using *vat* audio conferencing tool developed by Lawrence Berkeley Laboratory during the MBONE live report of NASA shuttle mission STS-74 on Nov. 20, 1995. In an observed interval of 10 minutes, playback delay was constantly adjusted by *vat* (in a range of 150 ms to 2 s). In such a unidirectional transmission case, a long playback delay is not a problem. The playback delay is usually set long enough so that usable packets are rarely dropped. Other observations have also been made. For example, audio packets are usually of constant length and posses a higher packet rate than video packets, which are usually of various length.

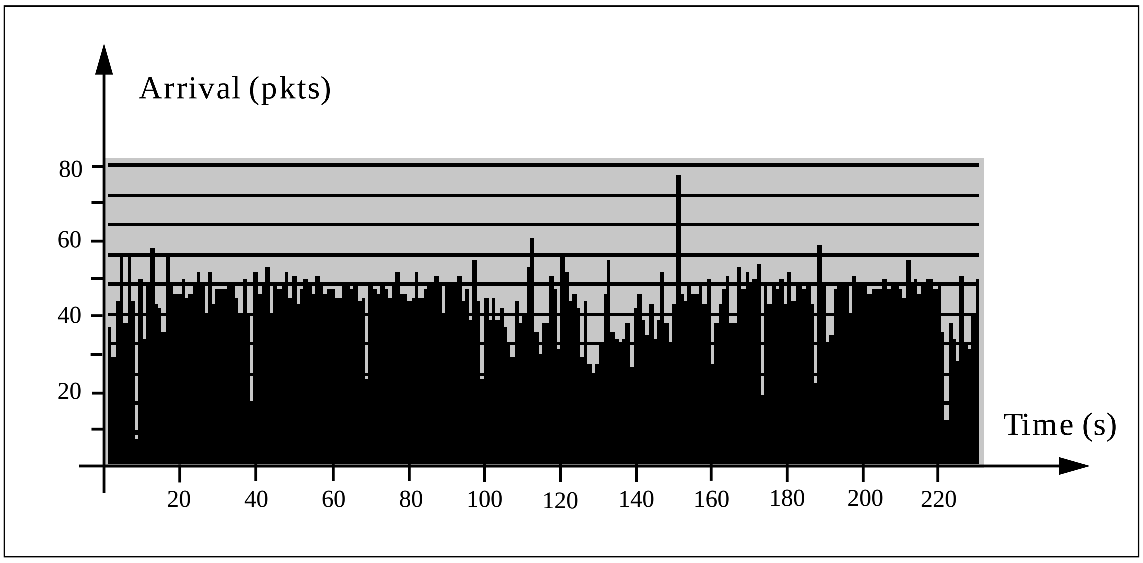

***Figure 6-1*** **Histogram of audio packet arrival over Internet**

The key features of the CM applications are embodied in the simulation in the form of play-back scenarios, as analyzed in *Section 3.2.1 Modeling CM activity as play-back application*. This is based on the fact that CM data are usually generated at the source periodically and continuously and are then consumed at a destination periodically and continuously. The use of playback delay can accommodate the irregular delays (delay jitters) of packets as well as out-of-order packets to the extent where they are still ready for their playback. In Figure 6-2, for example, the playback of the audio packets are delayed to 80 ms after the mean ready time. Packet no. 4 is out of order but is still ready in time for its playback. Packet no. 9 is too late and its place for playback might be replaced by a dummy audio sample instead.

Although it is a normal practice for the adaptive continuous media applications to adjust their playback delays according to the changing situations, the whole lifetime of an application can still be seen as a composition of a series of playback sections each with a constant playback delay.

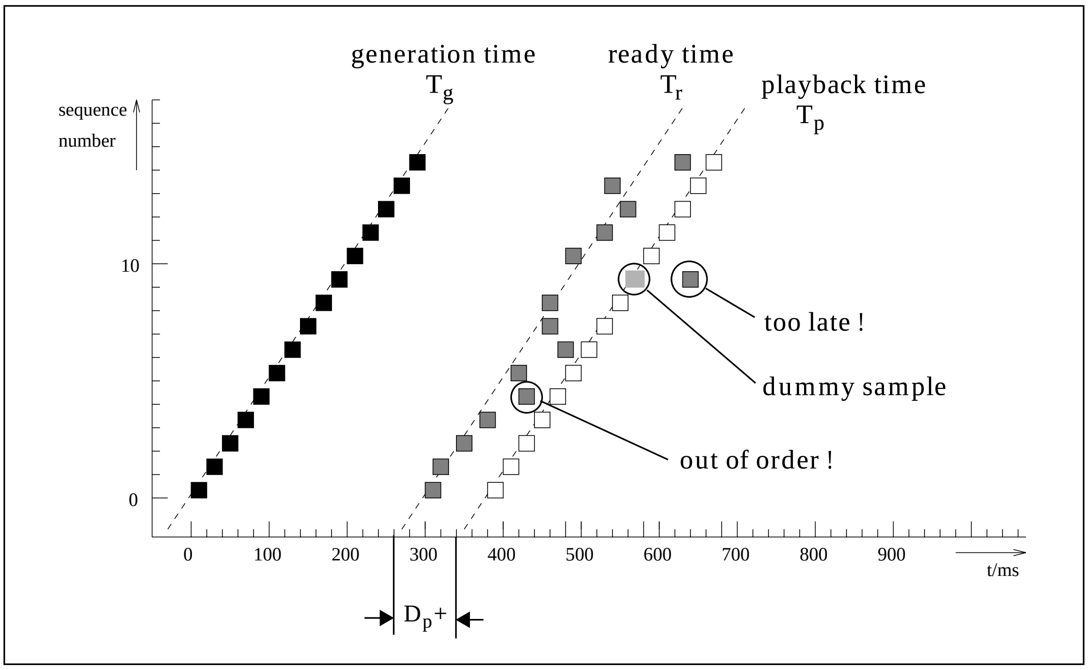

***Figure 6-2*** **Simulated audio playback**

The quality of the continuous media application depends mainly on the rates of packets which can be successfully played back. This is also the main index in the simulation. As explained in Chapter 3, a trade-off should be made to achieve the best total system effect. On the one hand, we have to keep the playback delay $D_{playback}$ small in order to reduce the negative effects induced by long end-to-end latency. On the other hand, we have to keep the playback delay $D_{playback}$ large enough in order to maintain a continuous playback by ensuring that most of the packets arrive by their playback time.

One interesting question is how to interpret the success rate for playback. For example, does a success rate below 90% means an unusable low quality? Actually, such question can only be answered in a concrete application environment. The loss tolerance of real-time audio transmission can usually vary in the range of 1% to 10% under different conditions. The conditions depend on factors such as coding methods, stream contents, loss distributions, etc. In some MBONE transmission, it has been observed that an audio stream is still comprehensible with a loss rate level at ~10%. Video stream transmission can tolerate even higher loss rate, provided that the human viewer is tolerant and patient enough.

The construction and parameterization of our simulation models have taken virtually all the above practical factors into consideration. In our simulation models, the parameters are somewhat abstracted. Delay is expressed in a kind of delay time unit. Computation time is also expressed in an abstract computation time unit. In this way, we are mostly independent of concrete communication links and processor types. Still, the simulation results can find their direct applications in practical scenarios.

The use of simulation has brought with it many flexibilities. For example, some complex process sets including A/V processes, virtual periodic non-RT processes, and other non-RT processes can be easily simulated with a wide range of parameter settings. It would usually be very difficult to arrange some such parameter settings in an implementation environment. The results from simulations can also be used in choosing the feasible algorithms to be implemented in our experimental systems.

## 6.2 Simulator and parameterization

The simulator is composed of several functional components with diverse parameter setting possibilities [Fan96b].

### 6.2.1 Structure of the simulator

In the simulator, the following functional components can be roughly identified:

- the component which embodies the various processes including real-time periodic processes for continuous media, virtual real-time periodic processes and other normal processes;
- the component which realizes different soft real-time scheduling and handling algorithms;
- the component which reflects the characteristics of the networking and pro-

cessing environment;

- and the house-keeper component which is responsible for initialization, monitoring and accounting of the simulation.

These components are to a large extent self-contained so that different mechanisms and algorithms can be used and extended in each of them.

In order to mimic a scheduling subsystem, the simulator has simulated a discrete tick-driven system in which processes have to complete their jobs. A job is an abstraction for data packets or other events which should be worked on by a process. From a dynamic point of view, as shown in Figure 6-3, a simulation run comprises the following four phases:

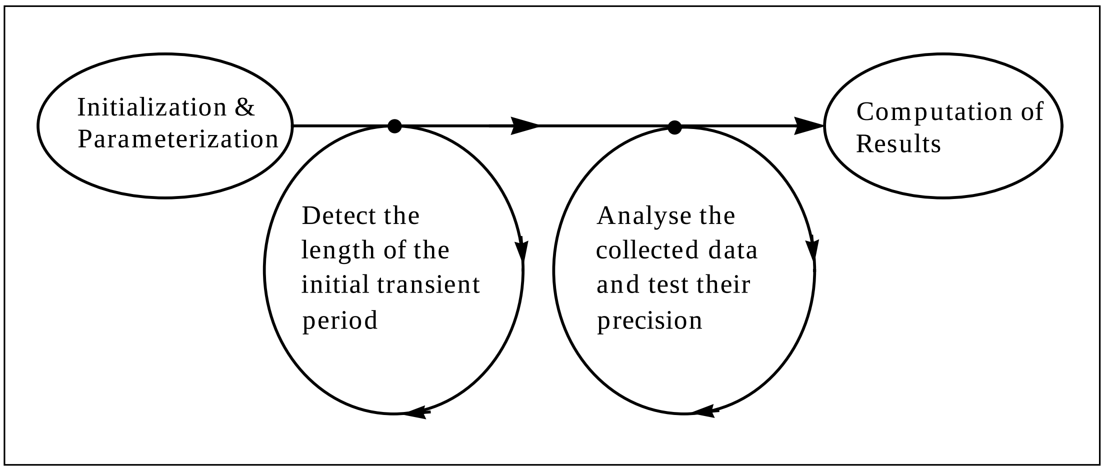
Initialization & Parameterization
Detect the length of the initial transient period
Analyse the collected data and test their precision
Computation of Results

***Figure 6-3*** **Four phases of a simulation run**

*Phase 1* — The components are initialized with corresponding parameters. Different running entities are created.

*Phase 2* — The simulated queueing system is checked to reached its statistical equilibrium. The determination is mainly with regard to the success rate of all processes. During this transient period, the simulation outputs are simply discarded.

*Phase 3* — This is the main part of a simulation run. In this phase, different simulation data are produced and collected. The simulation will be run until the required precisions of all the main target results are reached.

*Phase 4* — According to the collected sample data, the results are computed and formatted.

A suitable solution for deciding upon the starting point of a steady-state analysis in the phase 2 and a technique for obtaining the final simulation results to a required level of accuracy are both taken from [Pawlikowski90]. Concretely, we use the following rule to determine the initial transient period:

Rule 4 of [Pawlikowski90]: In a time series of observations $x_1$, $x_2$, ..., $x_i$, ..., the initial transient period is over after $n_0$ observations if k consecutive values of the running mean $\overline{X}(i)$ recorded after the observation $n_0$ differ less than 100δ% from $\overline{X}(n_0+k)$; that is, for all i, $n_0 < i <= n_0+k$,

$$\frac{\left|\overline{X}(n_0 + k) - \overline{X}(i)\right|}{\left|\overline{X}(n_0 + k)\right|} < \delta.$$

The initial parameters are set to: k = 100, δ = 0.01.

We use the method of non-overlapping batch means to determine when the required precision has been reached. The first step is to determine the batch size m* such that batch means are (almost) uncorrelated at a given level of significance. In the second step, the accuracy of the results are analyzed by calculating their confidence interval. The simulation is continued and more observations are collected to improve the accuracy of the results until the required precision has been reached. In the beginning, the batch size $m_0$ is set to 100 and the number of batch means $k_{b0}$ is set to 100.

The simulation results reported below all posses a confidence level of 99% (α=0.01) with a maximum acceptable relative precision of confidence interval of 1% ($\varepsilon_{max}$=0.01) with regard to the success rate of playback — which is the main target result. Other results are usually collected when the precision for the main target result has been reached.

### 6.2.2 Varying factors

A simulation run can be configured and parameterized in mainly three ways. The definition files for the programs of the simulator can be changed before their compilation. In this way, relatively stable parameters can be changed. For example, the parameters for detecting the initial transient period are given in this manner. The second way is to deliver a set of parameters to the simulator in a description file as the simulator is started. The description file contains, in addition to other things, a description of a set of the processes to be simulated. Still more parameters can be given to the simulator as command-line parameters.

The following lists some of the varying parameters which are necessary for the understanding of the simulation results to be presented:

- *ProcID* — Process identifier, a natural number.
- *Typ*e — "P" stands for a real-time periodical A/V process, "V" for a virtual real-time periodic process and "N" for a normal non-real-time process.
- *CompTime* — For real-time process, maximum processing time per period.
- *Period* — For real-time process, the period.
- *Quantum* — For non-real-time process, time quantum for one invocation.
- *Offset* — Start-time of a process.
- *JobCnt* — Maximum number of the jobs the process will try to complete in a period. For non-real-time process, a value of zero can be given to designate a "load" process which is always ready for consuming computation time.
- *BufferSize* — The size of the buffer for the jobs which have arrived and waited for processing.
- *JobOffset* — Time for the arrival of the first job for a process.
- *JobLoss* — Probability for the loss of a job.
- *JobSize* — Distribution of the size of jobs (in the sense of processing cost).
- *JobArrival* — Distribution of the arrival interval between jobs.
- *JobDelay* — Distribution of the variable part of the delay caused by "networking".
- *NetDelay* — Constant part of the delay caused by "networking".
- *PlaybackDelay+ (pbDelay+)* — Extra playback delay (End-to-end playback delay minus mean end-to-end delay which is *NetDelay* plus the mean of *Job-*

*Delay*).

For real-time A/V processes, the main target result is the rate of successful playback. Other results of interest include the number of timing violation, the variation of ready time, etc. For virtual real-time and non-real-time processes, the main target result is the achieved utilization. Several distribution functions have been used to parameterize *JobSize*, *JobArrival*, *JobDelay*, etc. In the following presentation, they are coded as a triple (e1, e2, e3) (or written as e1/e2/e3) for sake of brevity. The meanings of the elements are explained in the following table.

***Table 6-1*** **Coding for a distribution function**

| Functional description | e1 | e2 | e3 |
|---|---|---|---|
| Identity: constant as e2 | 0 | value | 0 |
| Uniform distribution | 1 | lower bound | upper bound |
| Exponential distribution | 2 | mean | 0 |
| Normal distribution | 3 | mean | variance |
| Distribution from trace | 5 | trace file | scaling factor |

### 6.2.3 Timing parameters

As is mentioned, the timing parameters are somewhat abstracted in our simulation models. Delay is expressed in a kind of delay time unit. Period and computation time are also expressed in an abstract computation time unit. In this way, we are mostly independent of concrete communication links and processor types. Still, a reasonable composition of relative parameters are necessary.

The main timing parameters of a soft real-time process used for multimedia related processing are its period and the computation time per period. While the period can usually be determined by the nature of the multimedia data to be processed and the number of samples to be processed pro period, it is relatively tricky to determine the per period computation time to be reserved. Processor types, data volume/complexity and processing paths vary. In addition, considerations must also be taken with regard to the timing-violation handling methods used and the degree of guarantee intended. These and other factors all have direct impact on the processing capacity needed.

In Figure 6-4, the decoding time for a MPEG stream is depicted. For an application like this, it is quite apparent that the processing cost varies quite considerably during the whole process. It is then a waste of resource if the CPU reservation is done according to the worst case.

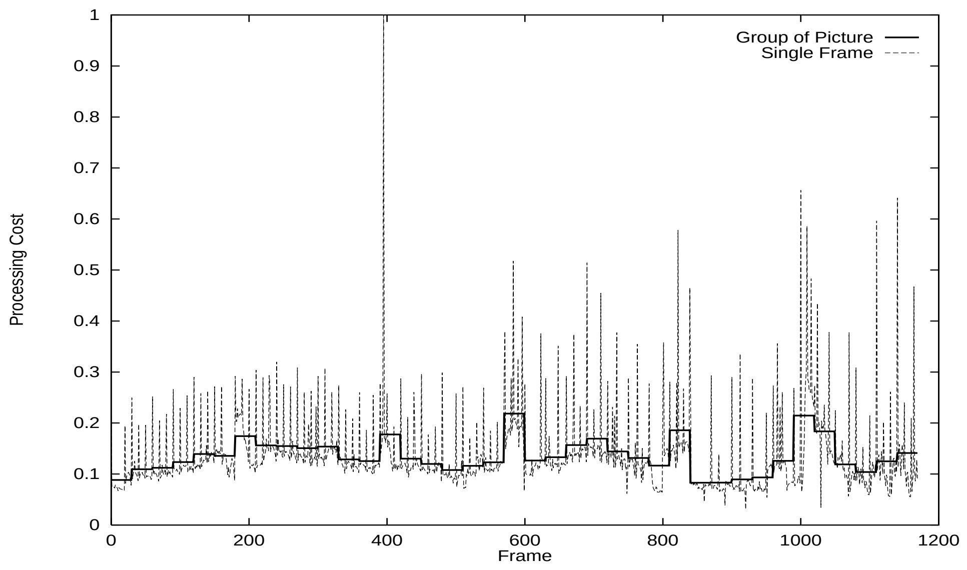

***Figure 6-4*** **Decoding time for a MPEG stream**

With our soft real-time approach, different degrees of optimism can be taken by the need of different degrees of soft guarantee. The timing handling methods for the cases of timing violations add another possible dimension of optimistic reservation of processing capacity. In can be anticipated, however, that it is usually infeasible to just reserve the mean actual processing time because of the variations in the actual processing time itself and the variations in job arrival.

If the mean actual processing need is larger than the reserved capacity, say actual_need = (100 + over)% * reserved_capacity. Then, as a upper bound, success_rate ≤ (100/(100 + over)). For over = 25, i.e. reservation_grade = 125%, success_rate ≤ 100/125. That is, success_rate ≤ 80%.

In the following simulations, various distributions are used for job computation-size and job arrival. The impact of various rates of mean actual job computation-size to reserved per-period computation time is also evaluated.

It remains to be mentioned that the simulation results are directly applicable to each working phase of a set of multimedia applications. Between working phases where timing parameters might vary considerably, the FAST adaptive service provision model of Chapter 9 should come to play a critical role.

## 6.3 Soft real-time effects and their analyses

We are now ready to present some of the simulation results and relate the simulation results to their practical meanings.

### 6.3.1 Soft real-time is necessary and better

The flexibility and adaptability of the multimedia applications only make their underlying real-time support “softer” and (generally) easier to implement. Some grade of real-time support is, however, necessary. Otherwise, either the quality of the MM functions can not be guaranteed to be satisfactory or resources could not be exploited to support as many applications as they could possibly be.

#### 6.3.1.1 Rate-monotonic vs. round-robin

Real-time scheduling can achieve the effect that the timing and execution characteristics of a process is virtually isolated from other processes thus providing a guarantee for its computation QoS. Simple round-robin scheduling would let the process take a smaller share if more processes compete for the computation capacity. In a simple scenario, for example, we have the following three simplified processes which should handle some A/V related work:

| *ProcID* | *CompTime* | *Period* | *Offset* | *JobSize* | *JobArrival* | *JobDelay* | *pbDelay+* |
|---|---|---|---|---|---|---|---|
| *1* | *3* | *11* | *83* | *0/3/0* | *0/11/0* | *0/5/0* | *18* |
| *2* | *7* | *31* | *101* | *0/7/0* | *0/31/0* | *0/5/0* | *38* |
| *3* | *9* | *47* | *97* | *0/9/0* | *0/47/0* | *0/5/0* | *54* |

In addition to these processes, which account for a total usage of 0.69, there are a number of load processes which each consumes a time slice of 1 time unit in each invocation.

Under rate-monotonic scheduling, the above A/V processes are assigned priorities according to the lengths of their periods. The load processes are assigned a lower priority as in the SIM-SCHEDULE or CO-SCHEDULE scheduling algorithms given in *Section 5.2 Proposed soft real-time scheduling schemes*. Under round-robin scheduling algorithms, the A/V processes are each given a time slice of their respective *CompTime*'s at each invocation. The success rate for playback is clearly dependent on the scheduling method used. While the rate-monotonic scheduling algorithm has guaranteed the usage of the three A/V processes, the simple round-robin scheduling has lead to a unusable low quality for them, especially under heavy load. See Table 6-2 for some numerical examples.

***Table 6-2*** **Dependence of success rate on scheduling methods**

| Success rate in % | Rate-Monotonic | | | Round-Robin | | |
|---|---|---|---|---|---|---|
| Number of load processes | P1 | P2 | P3 | P1 | P2 | P3 |
| 0 | 99.9+ | 99.9+ | 99.9+ | 99.9+ | 99.9+ | 99.9+ |
| 1 | 99.9+ | 99.9+ | 99.9+ | 95.31 | 99.98 | 99.97 |
| 2 | 99.9+ | 99.9+ | 99.9+ | 67.32 | 99.98 | 99.97 |
| 3 | 99.9+ | 99.9+ | 99.9+ | 36.18 | 99.88 | 99.97 |
| 4 | 99.9+ | 99.9+ | 99.9+ | 18.10 | 93.75 | 99.97 |
| 5 | 99.9+ | 99.9+ | 99.9+ | 8.76 | 93.23 | 97.36 |
| 10 | 99.9+ | 99.9+ | 99.9+ | 1.45 | 41.89 | 95.96 |
| 20 | 99.9+ | 99.9+ | 99.9+ | 0.96 | 1.59 | 36.49 |

#### 6.3.1.2 More applications feasible by soft real-time

As a result of real-time scheduling and timing enforcement, more concurrent multimedia applications can be supported satisfactorily in a soft real-time environment than in a time-sharing environment.

Suppose a set of video processes should be supported in a video-on-demand server. The success rates for these processes should be, of course, very high. The following simulates the situation in a SIM-SCHEDULE system and a round-robin system. The video processes are assumed to posses the following parameters:

*compTime* 4, *Period* 40, *Offset* 100, *JobCnt* 3, *Buffersize* 100, *JobOffset* 92, *JobLoss* 0, *JobSize* 1/3.0/3.9, *JobArrival* 1/39.5/40.5, *JobDelay* 0/10/0, *playbackDelay+* 150.

That is, the job size and arrival of the video samples are quite regular (uniform distributions in small ranges). It is further assumed that there are some small load processes each consumes a time slice if it is allowed to run. The following table gives the success rate for the SIM-SCHEDULE system and the round-robin system.

***Table 6-3*** **Feasible concurrent applications and scheduling methods**

| Success rate in % | Number of concurrent instances of applications | | | | | | |
|---|---|---|---|---|---|---|---|
| Number of small load processes | SIM-SCHEDULE | Round-Robin | | | | | |
| | 10 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 99.99+ | 99.99+ | 99.99+ | 99.99+ | 99.99+ | 99.99+ | 99.99+ |
| 5 | 99.99+ | 99.99+ | 99.99+ | 99.99 | 99.99 | 99.99 | 99.99 |
| 10 | 99.99+ | 99.99 | 99.99 | 99.98 | 99.89 | 96.65 | 28.05 |
| 15 | 99.99+ | 99.99 | 99.97 | 99.85 | 47.34 | 25.63 | 18.89 |
| 20 | 99.99+ | 99.97 | 99.40 | 36.38 | 23.33 | 17.75 | 14.05 |

Since the video processes are harmonic, the SIM-SCHEDULE system can support up to 10 instances of the video process concurrently (with an admissible total capacity of 1). The success rate for these processes are always higher than 99.9%, no matter how many load processes there are. In contrast, the video processes in the round-robin system are disturbed by those load processes. If a threshold for 99% success rate is required, then the heavily shaded area is not usable. For example, only up to 7 instances of video process can be supported by round-robin with 15 small load processes. If a higher threshold success rate of 99.9% is set, then only up to 5 instances of video process can be supported by round-robin with 20 small load processes (lightly shaded area also not usable).

### 6.3.1.3 Isolation of timing characteristics

Strict hard real-time scheduling try to guarantee the timing requirements of the hard real-time processes within the range of their worst run-time behavior. It can thus achieve the effect of isolating one hard real-time process from the timing and execution characteristics of other processes in the similar way that a memory protection system isolates it from outside memory access. The resource allocation and the process model for hard real-time system are thus very constrained in order to make the guarantee even in the worst possible scenarios. We have argued that more relaxed soft real-time models are applicable for multimedia applications.

All of the scenarios simulated here are soft real-time in the following aspects: (1) no 100% success rate is taken as a goal; (2) not strict rate-monotonic model where each process should be ready at the beginning of a period; (3) loose interdependence between processes can be ignored; (4) optimistic admission control allowed, especially for maybe-guarantee; etc. Even with these relaxations, it can be seen from the numerous simulation results that the predictability features of real-time scheduling are well reserved through the use of soft real-time scheduling and handling schemes.

The following example should show that soft real-time can ensure the timing feature of A / V processes even in a competitive situation. The SIM-SCHEDULE is used to schedule the processes and the RELAY method is used to handle timing overflow. The following parameters for audio and video processes are used:

```
ProcID CompTime  Period   Offset JobCnt   JobSize   JobArrival  JobDelay
Audio       2        20       100     3     2/1.5/0    0/20/0      2/50/0
Video       4        40       100     3     2/3.0/0    0/40/0      2/50/0
```

In the first simulation, only one audio process and one video process is run. In the second simulation, three audio processes and three video processes are run concurrently. We note that the total CPU usage of them (0.6) is still within the schedulability bound (0.74 for 6 RT processes, see Table 4-1).

Figure 6-5 shows the results for running one pair of A / V processes separately and Figure 6-6 shows the results for running three pairs of A / V processes concurrently. Table 6-4 has given some numerical results for a better inspection.

**Table 6-4 Separate and concurrent running of A/V processes**

| Success rate in % | Separate running | | Concurrent running | | | | | |
|---|---|---|---|---|---|---|---|---|
| Playback delay+ | A | V | A1 | A2 | A3 | V1 | V2 | V3 |
| 40 | 80.5 | 62.9 | 87.0 | 86.5 | 80.5 | 63.4 | 63.4 | 63.6 |
| 50 | 91.1 | 84.3 | 94.8 | 94.4 | 91.1 | 69.1 | 77.6 | 85.6 |
| 60 | 89.9 | 84.2 | 94.4 | 93.8 | 90.1 | 85.1 | 84.4 | 84.9 |
| 80 | 94.8 | 83.8 | 97.7 | 97.1 | 94.8 | 84.4 | 84.2 | 85.0 |
| 100 | 97.1 | 92.9 | 98.8 | 98.7 | 97.1 | 93.6 | 93.1 | 93.7 |
| 150 | 99.6 | 96.6 | 99.9 | 99.8 | 99.6 | 97.1 | 96.7 | 97.1 |
| 200 | 99.8 | 98.4 | 99.9+ | 99.9 | 99.8 | 98.6 | 98.3 | 98.5 |
| 250 | 99.9+ | 99.6 | 99.9+ | 99.9+ | 99.9+ | 99.4 | 99.5 | 99.6 |

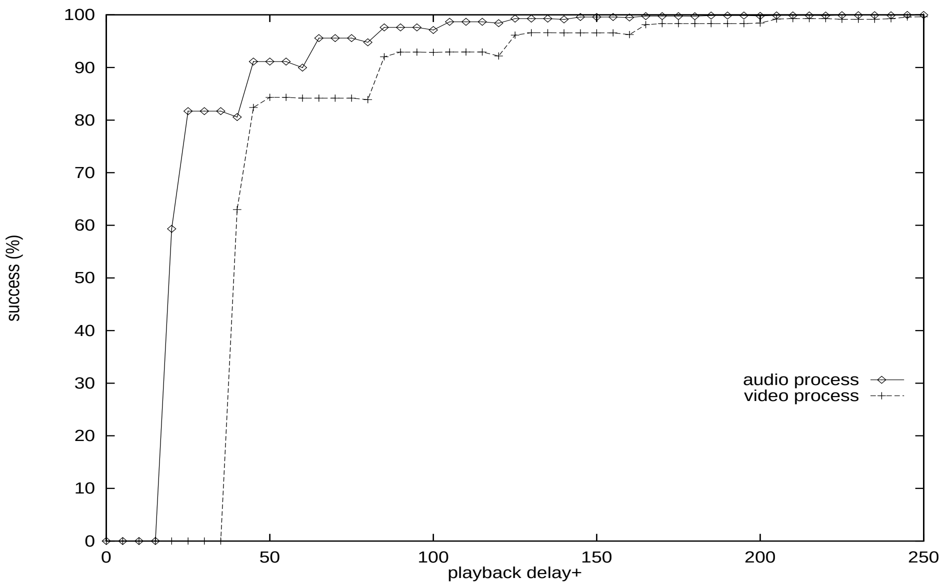

***Figure 6-5*** **Success rate for separate running of A/V processes**

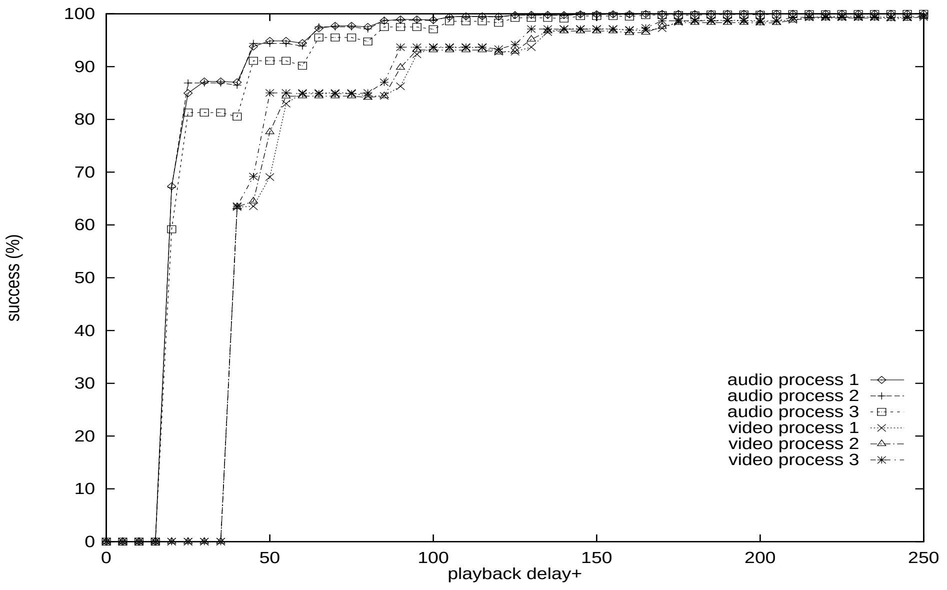

***Figure 6-6*** **Success rate for concurrent running of A/V processes**

As can be seen from the figures and the table, even in the case of concurrent competitive execution of several A/V processes, their timing requirements are still well met in the sense that they are not influenced negatively by the competitive processes. The timing results are similar to the case where they are executed separately. The timing specification of each process is fulfilled without sacrificing the timing reservations of other processes. A careful reader might notice the small differences among the success rates of the concurrent audio processes or video processes. These small differences have resulted from the fact that three instances of the audio process and the video process are now running. Their executions should be scheduled sequentially although their periods are identical. Since the play-back delays are not always just at the end of a period, some small differences in the success rates of the three instances are inevitable. Such differences are smoothed out by large playback delays.

Since soft real-time can also achieve the effect of isolating one soft real-time process from the timing and execution characteristics of other processes to a large extent, the simulation results of a single set of real-time processes can as well be scaled up to apply to several concurrent and similar sets of real-time processes as long as their usage capacities are still in the schedulability bound. This effect has lead to some simplifications in the simulations of process sets and the presentations of their results in the following.

#### 6.3.1.4 Regularity and synchronism

Soft real-time can make sure that an A/V process can take its share of execution and thus make progress regularly according to its timing reservation. This is a good feature for the concurrent reservation of other resources such as buffers and I/O devices. The feature is also well suited for the cases where some forms of synchronism are needed. The feature of synchronism is quite important for such cases as the synchronization between the playbacks of related audio and video streams (lip-synchronization), where a deviation of 50 ms can already be clearly noticed and be considered as disturbing.

The following investigates a case for the parallel playbacks of an audio stream and a video stream. It is assumed that the A/V packets have been processed by other processes and are put into some kinds of ring-buffer. Here, the attention is paid to the processes which are directly responsible for retrieving A/V samples from the ring-buffer to A/V devices for playback. For them, the job size and job arrival are normally quite regular, as is experienced in some software implementation such as NeVoT:

```
ProcID CompTime  Period   Offset JobOffset  JobSize   JobArrival
Audio      6        20       100     90        1/5.8/6    0/20/0
Video     12        40       100     90      1/11.8/12    0/40/0
```

The system is assumed to be loaded additionally by non-real-time load processes each with a quantum of 1.

The following figures show a section of synchronism between the corresponding playback points of audio and video samples under different degrees of system loads. As also can be seen in the numerical examples in Table 6-5, the synchronism is much worse under round-robin scheduling than under CO-SCHEDULE scheduling, especially under heavy loads. For sake of comparison, the y-range of Figure 6-8, Figure 6-9 and Figure 6-10 are all set to [0, 100].

***Table 6-5*** **Synchronism between playback points of A/V processes**

| Scheduling method | synchronism in time unit | | |
|---|---|---|---|
| | Maximum | Mean | Standard deviation |
| CO-SCHEDULE | 1.95 | 1.82 | 0.0714 |
| Round-Robin 0%-Load | 2.45 | 1.56 | 0.4889 |
| Round-Robin 10% Load | 16.6 | 4.93 | 3.9092 |
| Round-Robin 20% Load | 53.0 | 9,42 | 8.5102 |
| Round-Robin 30% Load | 90.7 | 17.5 | 15.5831 |

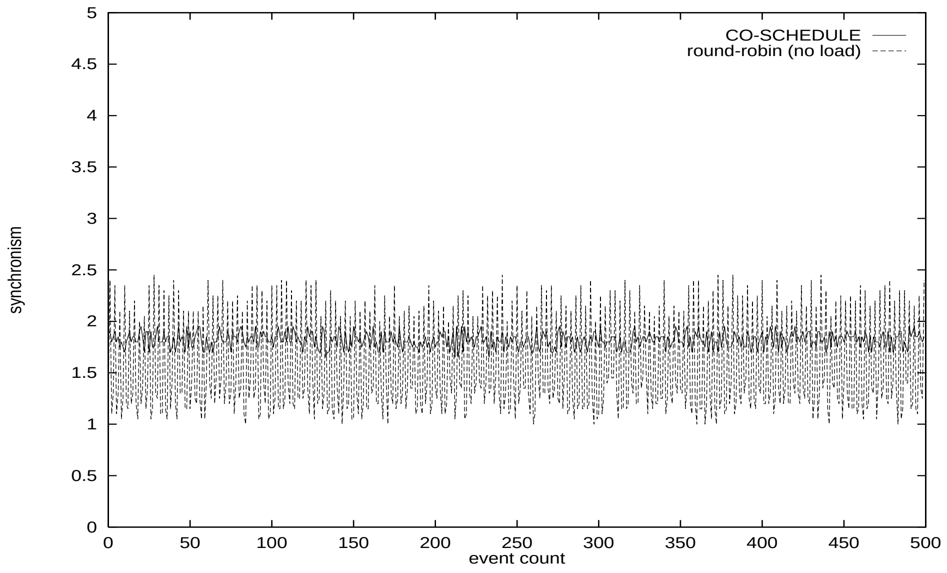

***Figure 6-7*** **Synchronism of playback points with no extra load**

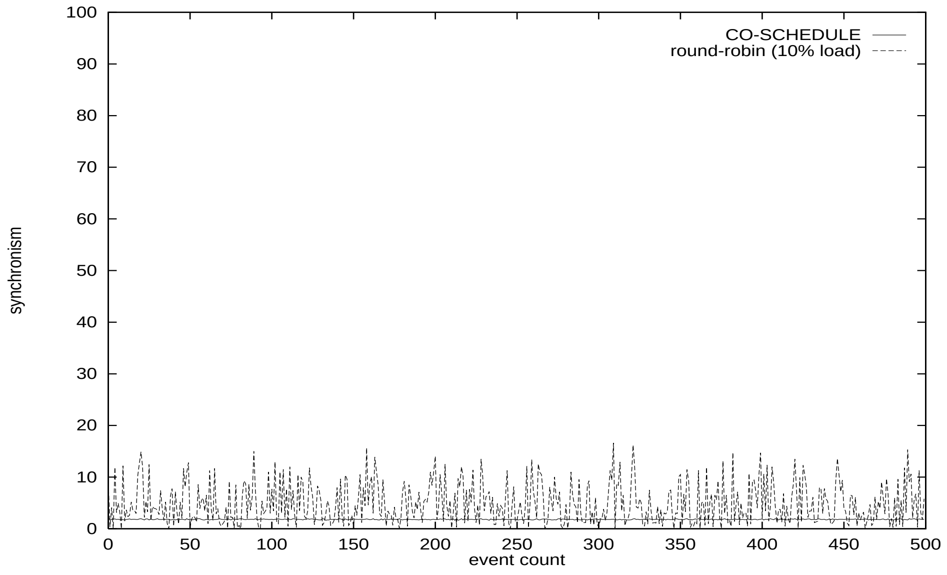

***Figure 6-8*** **Synchronism of playback points with 10% system load**

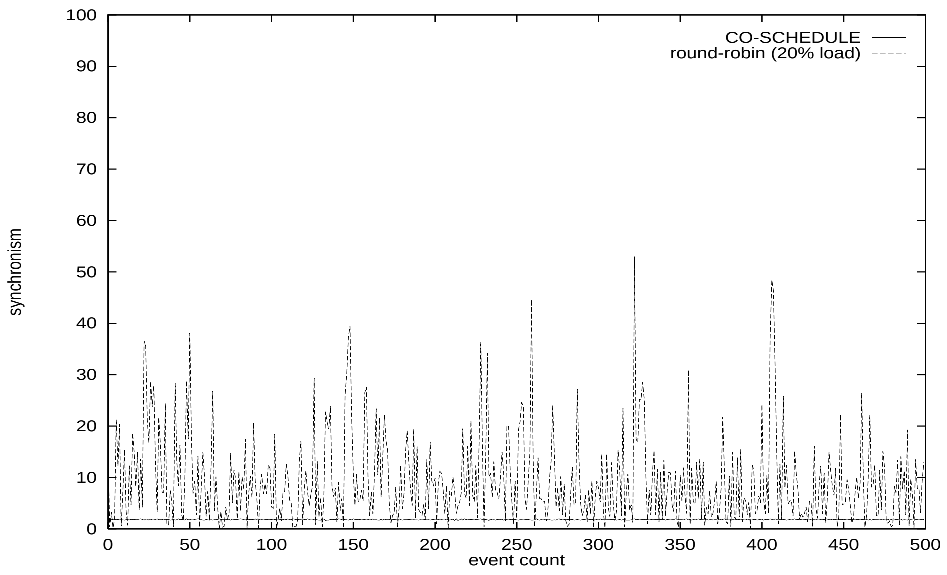

***Figure 6-9*** **Synchronism of playback points with 20% system load**

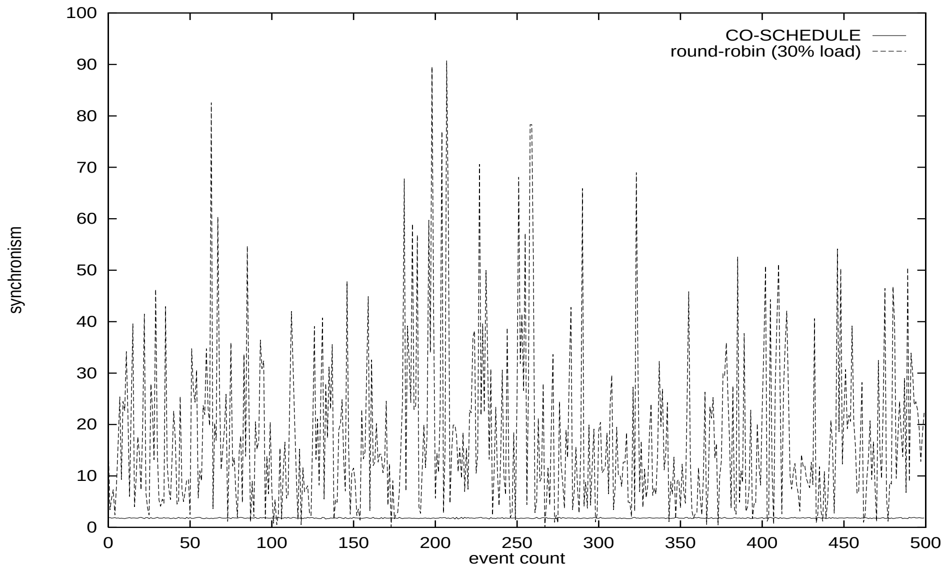

***Figure 6-10*** **Synchronism of playback points with 30% system load**

### 6.3.2 Pros and cons for monitoring and enforcing

In Section 6.3.1.3, a scenario under SIM-SCHEDULE has been simulated where several processes run concurrently. The effects are virtually the same as the processes are run separately and alone. That means, their executions are quite independent from one another. This achieves the effect that the timing and execution characteristics of the concurrent processes are more or less isolated from one another. Thus SIM-SCHEDULE has reduced the negative influence of timing violations of one process on other processes to a minimum and has thus guaranteed their respective processing capacity reservations.

The following simulation should further contrast the pros and cons of timing monitoring and enforcing in a scheduling system. The following process sets are scheduled under CO-SCHEDULE and SIM-SCHEDULE respectively:

| *ProcID* | *Type* | *CompTime* | *Period* | *JobSize* | *JobArrival* | *JobDelay* | *pbDelay+* |
|---|---|---|---|---|---|---|---|
| *10* | *P* | *4* | *20* | *2/3.0/0* | *1/19.5/20.5* | *1/14.9/15.1* | *120* |
| *11* | *P* | *8* | *40* | *2/6.0/0* | *1/39.5/40.5* | *1/14.9/15.1* | *200* |
| *12* | *P* | *14* | *80* | *2/10.5/0* | *1/79.5/80.5* | *1/14.9/15.1* | *350* |
| *13* | *P* | *16* | *80* | *2/12.0/0* | *1/79.5/80.5* | *1/14.9/15.1* | *350* |
| *14* | *P* | *10* | *80* | *2/7.5/0* | *1/79.5/80.5* | *1/14.9/15.1* | *350* |
| *15* | *V* | *40* | *400* | *0/39.9/0* | *0/400.0/0* | *0/2.0/0* | *2000* |

Other parameters for these processes are less relevant and are set to the same. For example, `JobCnt=4, BufferSize=50, JobLoss=0, NetDelay=10.` Since this is a harmonic process set, it is schedulable under the rate-monotonic scheme (with the total capacity less or equal to 1).

Process 12, 13, 14 have the same length of period. According to the extended rate monotonic theory, their priority orders among themselves can be arbitrary (thus breaking ties). It is assumed that the system gives them their priority orders in the order as listed above. Process 15 is a virtual-periodic real-time process which needs 40 time quantums in each period of 400. In a normal situation, all processes receive a workload of 75% as listed above. For other situations, however, it is assumed that the work load for process 11 is heavier than expected. In overload situation 1, the `JobSize` for process 11 is assumed to be `2/12.0/0`, i.e., 150% of the declared (expected) capacity. In situation 2 and 3, the values are `2/18.0/0` (225%) and `2/24.0/0` (300%) respectively.

The following table shows the success rates of all real-time processes and the actual usage of the virtual real-time process. Note again that the CO-SCHEDULE system schedules the processes in the rate-monotonic manner but does no timing monitoring and enforcing. The SIM-SCHEDULE system not only schedules the processes in the rate-monotonic manner but also monitors their usages constantly. In the case of timing violations, the RELAY handling method is used here to achieve timing enforcement.

***Table 6-6*** **Effects concerning timing monitoring and enforcing**

| Success rate in % | CO-SCHEDULE | | | | SIM-SCHEDULE | | | |
|---|---|---|---|---|---|---|---|---|
| | Situation | | | | Situation | | | |
| ProcessID | Normal | Over-load 1 | Over-load 2 | Over-load 3 | Normal | Over load 1 | Over load 2 | Over load 3 |
| 10 | 99.99+ | 99.99+ | 99.99+ | 99.99+ | 98.39 | 98.39 | 98.39 | 98.39 |
| 11 | 99.99+ | 99.99+ | 99.85 | 97.63 | 97.02 | 34.50 | 13.30 | 6.93 |
| 12 | 99.99+ | 99.97 | 97.58 | 81.08 | 96.03 | 96.04 | 96.07 | 96.04 |
| 13 | 99.99 | 99.02 | 85.79 | 52.37 | 96.65 | 96.41 | 96.41 | 96.41 |
| 14 | 99.71 | 92.69 | 61.18 | 24.81 | 96.70 | 96.45 | 96.45 | 96.45 |
| 15 (usage) | 9.98% | 9.97% | 6.19% | 1.69% | 9.92% | 9.92% | 9.92% | 9.92% |

Under CO-SCHEDULE, Process 11 maintains a high success rate at the cost of other processes. The overload of process 11 has direct negative influences on all the processes (process 12 - 15) which have lower priorities as arranged by the rate-monotonic scheme. The heavier the overload is, the worse the negative influences are. In overload situation 2, for example, the success rates for process 13 and 14 are only 85.79% and 61.18%. Process 15 only gets a usage of 6.19%, far less than reserved.

Under SIM-SCHEDULE, the overload of process 11 has only a direct negative influence on itself. All other processes are insulated from its overload. In overload situation 2, for example, the success rate for process 11 is down to 13.30% but the success rates for process 13 and 14 are as in the normal situation (96.41% and 96.45% respectively). Process 15 also gets a normal usage as reserved.

Generally, timing monitoring and enforcement is not important in a static, constrained and stable system environment. In such an environment, a scheduling system such as CO-SCHEDULE has its advantage of simplicity and efficiency. Transient and slight overloads can be more or less compensated between processes silently.

However, timing monitoring and enforcement is important for general and dynamic system environments. In such environments, overloads are unavoidable and common due to many reasons. The overload of process 11, for example, might be caused by over-optimism (much lower load expected), selfishness (don't want to "pay" as much) or malice (don't care for the system and other users) of the process creator. Anyway, a scheduling system like SIM-SCHEDULE can monitor the execution of the system and take enforcement measures in case of need. The reservations and guarantees of timing characteristics of the whole system can thus be achieved in a predictive and stable manner. In the following, SIM-SCHEDULE will be the main emphasis of further investigations.

### 6.3.3 Effects of soft real-time handling of timing-overflow

As pointed out, a scheduling method like SIM-SCHEDULE can reduce the negative influence of timing violations of one process on other processes to a minimum and can thus guarantee their processing capacity reservations to a large extent. For each process itself, soft real-time handling for timing-overflow is still necessary since soft real-time allow occasional timing violations both from the side of the system and from the side of user processes.

The following simulations should show the effects of different soft real-time handling methods. The simulations are mainly varied with regard to different lengths of play-back delays and different degrees of reservation optimism, i.e., different rates of mean actual computation time to reserved computation time. The process and its variations are parameterized as follows: *compTime* 10, *Period* 20, *Offset* 100, *JobCnt* 1-3, *Buffersize* 100, *JobOffset* 0, *JobLoss* 0, *JobSize* 50%(2 / 5 / 0) | 75%(2 / 7.5 / 0) | 100%(2 / 10 / 0) | 125%(2 / 12.5 / 0), *JobArrival* 0 / 20 / 0, *JobDelay* 0 / 50 / 0, *netDelay* 50, *playbackDelay+* 0 ...... 800 (step 5). That is, the reserved computation time is 10 time units. The actual computation capacity needs are respectively 50%, 75%, 100% and 125% of the reserved computation capacity. The computation sizes of jobs are exponentially distributed and playback delay plus is varied from 0 to 800 with a step of 5 time units.

From the following figures, it can be seen that the methods of CANCEL, RELAY, RELAY-TRUNC from *Section 5.3 Proposed soft real-time handling methods* have achieved different system effects. The curves are parameterized by the reservation grade, i.e., by the ratio of the mean actually required computation time to the reserved computation time.

The effects of handling timing-overflow by the CANCEL method are shown in Figure 6-11. It shows that the success rates are generally not satisfactory in our case where the job sizes vary quite considerably. The success rate can not rise anymore after a certain length of playback delay. The reason is that only the jobs whose computation sizes are smaller than the reserved capacity in a period have the chances of being processed successfully. Computationally larger jobs will be canceled (aborted) anyway. A much longer setting of playback delay will not change this situation.

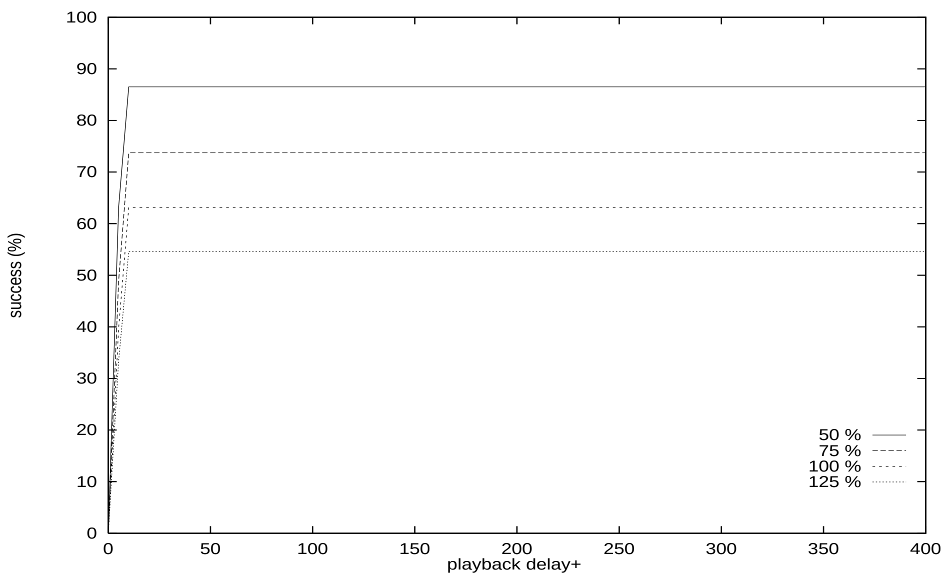

***Figure 6-11*** **Handling of timing-overflow with CANCEL**

Figure 6-12 shows the effects of handling timing-overflow by the RELAY method. A computationally large job will be relayed to the next period to be further processed if it can not be fully worked out in one period and thus has caused timing-overflow. Larger jobs and smaller jobs have thus the chances of compensate with each other to achieve a kind of inter-period compensation. The success rate becomes better generally with the increase of the playback delay.

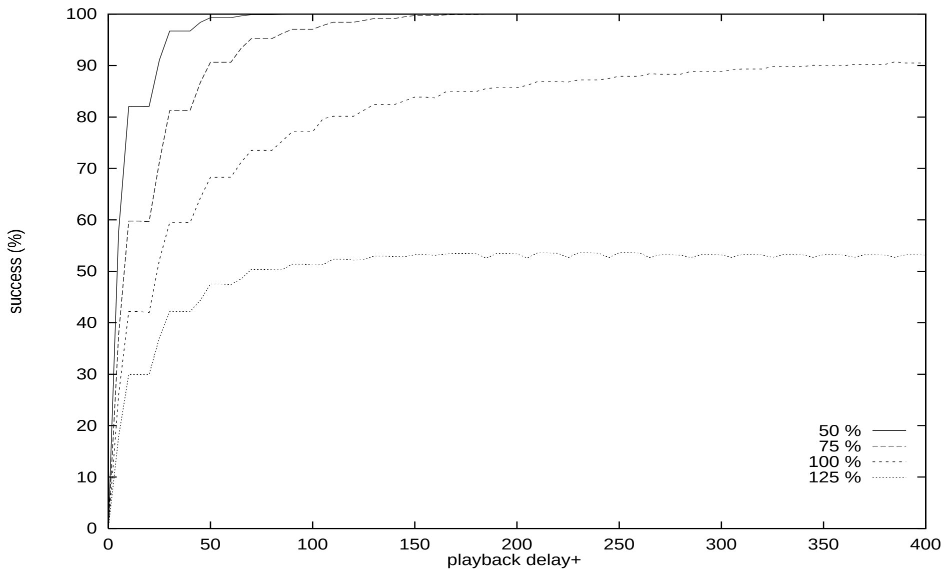

***Figure 6-12*** **Handling of timing-overflow with RELAY**

Due to the variation of the job sizes (modeled as an exponential distribution), the timing violations occur quite often in the above simulations. With a playback delay plus of 105 time units for 50% reservation grade (205 time units for 75% reservation grade, respectively), a success rate of 100% has been achieved for the RELAY method. The percentages of timing violations are 13.5% for 50%-curve and 26.3% for 75%-curve. For even more optimistic reservations, the timing violation rates are even higher. At a playback delay plus of 400, the violation rates are 36.9% for the 100%-curve and 45.4% for the 125%-curve. The violation rates are almost the same for the cases where the CANCEL method is used, where the violating jobs are simply discarded.

The above simulation concerning the RELAY method has been conducted by allowing the process to try to compute more than one job per period (`JobCnt=3`). If the process is only allowed to complete a job in a period, the effect of inter-period compensation would be almost useless. And the results would be very similar to the CANCEL method (see Figure 6-13).

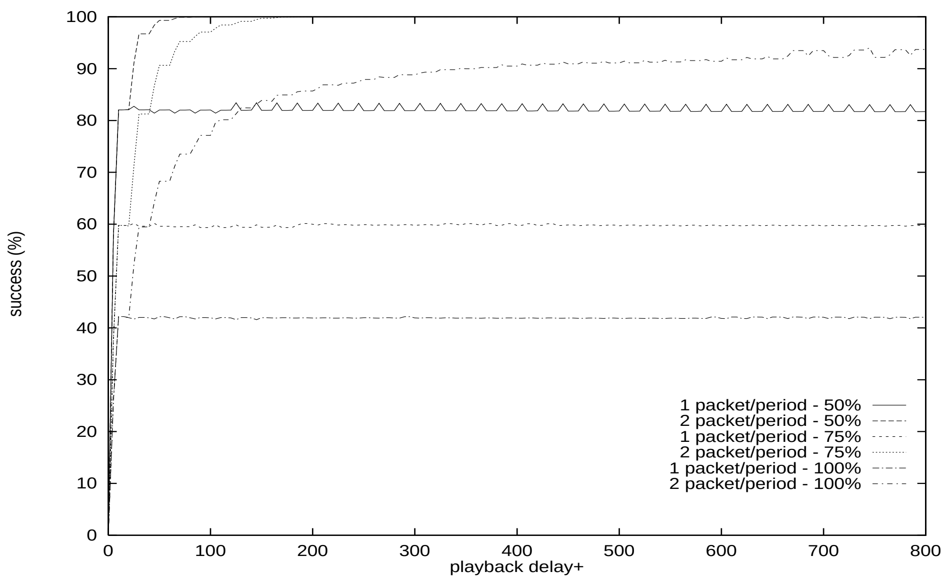

***Figure 6-13*** **Effects of inter-period compensation by RELAY**

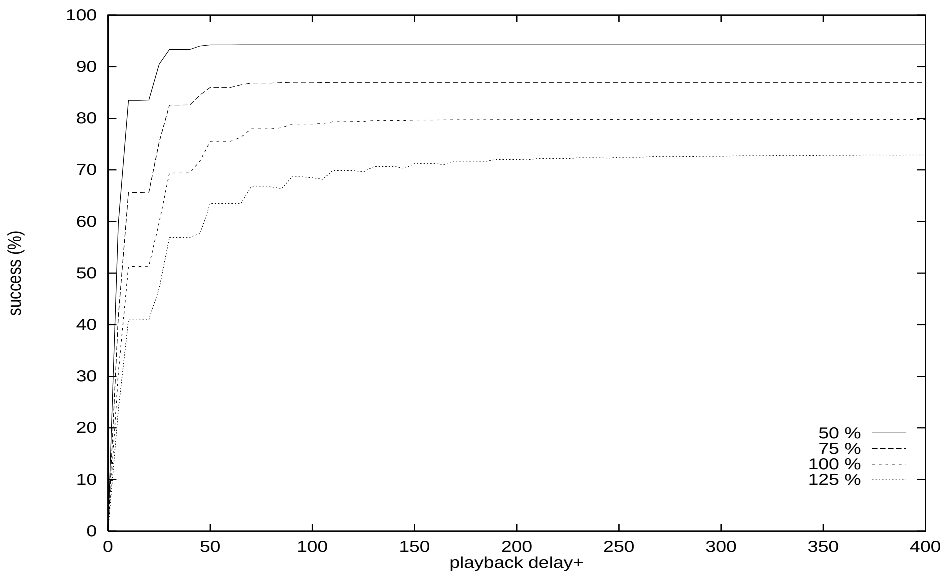

***Figure 6-14*** **Handling of timing-overflow with RELAY-TRUNC**

A further simulation has dealt with the RELAY-TRUNC method. Here, a truncation is made if a job still can not be completed after it is relayed once. The results are shown in Figure 6-14. The effects of combining the elements of RELAY and CANCEl can be seen quite clearly. For more conservative reservations, the success rates of RELAY-TRUNC can not be as high as RELAY, even if the playback delay has been set quite long. The reason is that a certain percentage of large jobs have been truncated after being relayed once. On the other hand, the success rates for RELAY-TRUNC are better than the case of RELAY in over-optimistic case. The 125%-curve, for example, is clearly better than its RELAY counterpart.

Some numerical examples taken from the above figures are compared in Table 6-7 to show the different influences of the handling methods for timing-overflow.

***Table 6-7*** **Influences of different handling methods for timing-overflow**

| Success rate in % | CANCEL | | | RELAY | | | RELAY-TRUNC | | |
|---|---|---|---|---|---|---|---|---|---|
| Playback delay+ | Reservation grade | | | Reservation grade | | | Reservation grade | | |
| | 75% | 100% | 125% | 75% | 100% | 125% | 75% | 100% | 125% |
| 5 | 48.8 | 39.6 | 33.3 | 37.6 | 25.9 | 17.8 | 41.6 | 30.6 | 23.4 |
| 10 | 73.7 | 63.1 | 54.6 | 59.7 | 42.2 | 30.0 | 65.6 | 51.3 | 40.9 |
| 25 | 73.7 | 63.1 | 54.6 | 71.3 | 52.2 | 37.1 | 75.4 | 59.8 | 47.0 |
| 50 | 73.3 | 63.1 | 54.6 | 97.8 | 68.3 | 47.5 | 86.0 | 75.5 | 63.5 |
| 100 | 73.3 | 63.1 | 54.6 | 99.9 | 77.1 | 51.2 | 86.9 | 78.8 | 68.5 |
| 200 | 73.3 | 63.1 | 54.6 | 100.0 | 85.7 | 53.4 | 86.9 | 79.7 | 72.0 |
| 400 | 73.3 | 63.1 | 54.6 | 100.0 | 90.5 | 53.2 | 86.9 | 79.7 | 72.8 |

Numerous other simulations with more varying factors (such as more variations in job arrival and job delay) have also been conducted to compare the influences of the different soft real-time handling methods for timing-overflow. The results are generally quite similar to the above situation. (See, for example, the next section — Section 6.4.) The RELAY and RELAY-TRUNC methods can usually achieve better success rates as long as the reservations are not made too optimistically. Another important point is that a larger job is sometimes also a more important job in many cases. For example, the I-frames in a MPEG-stream are usually the larger (and of course more important) frames than the P- and B-frames. It is therefore unwise to truncate the processing of such larger jobs. In addition, the truncations caused by the CANCEL and RELAY-TRUNC methods will inevitably incur the processing of inconsistent states from time to time. The RELAY method is, in contrast, more easy to use and will not cause complications in the applications. In this sense, the

RELAY method is the best choice for application if care has been taken to reserve sufficient processing capacity.

## 6.4 Validations with trace-oriented simulations

Most of the varying factors such as variations in job sizes and network delays have been modeled in the above simulations by some kinds of mathematical distributions such as uniform, exponential, normal distributions or their combinations. In addition to these “theoretical” simulations, we have also conducted some simulations based on the measurements from the real-world scenarios. If the results of the theoretical simulations and the trace-oriented simulations match well, our simulation models and their constructions will then be further validated. The following shows that this is indeed the case.

### 6.4.1 Measurements from real world scenarios

It is a well known problem that the processing capacity for an audiovisual stream varies from samples to samples, from packets to packets. In Figure 6-4 on page 137, the decoding time for a series of MPEG frames has been depicted.

For that measurement, the MPEG player *mpeg_play* from UC-Berkeley was run on a Sun Sparcstation 5/110 to decode a MPEG film named *ReadsNightmare.mpg* composed at the University of Erlangen-Nürnberg. The displays of the decoded frames were deactivated (with option `-no_display`) so that the processing cost was purely the cost for decoding. The film contains I frames, P frames and B frames. It is logically divided into a series of Group of Pictures (GoP) which is mostly of the bitstream order `IBBBBBBBBBPBBBBBBBBBPBBBBBBBBB` and which is mostly displayed in the order `BBBBBBBBBIBBBBBBBBBPBBBBBBBBBP`.

Two kinds of traces were recorded. One (`mpeg_frame.trace`) recorded the series of processing cost for each frames and another one (`mpeg_gop.trace`) recorded the series of processing cost for GoPs. The costs are scaled into the range of [0, 1] and are depicted as two separate curves in Figure 6-4. It can be seen that the processing cost for single frame varies quite considerably. In contrast, the processing cost for

GoPs stays relatively stable. The mean cost for single frame is computed to be 0.135 in `mpeg_frame.trace`. And the mean cost for Group of Pictures is 0.382 in `mpeg_gop.trace`.

The measurements of the processing cost of other audiovisual streams have shown similar appearances. In the following, `mpeg_frame.trace` is used as a representative for audiovisual streams with greatly varying processing cost. Other audiovisual streams such as a video stream consisting of JPEG frames might not vary so considerably. And `mpeg_gop.trace` is used as a representative for such streams with relatively stable processing cost.

It is also difficult to model the actual network traffic with some simple mathematical distributions. Network delays over LAN and WAN have been traced for the use of our simulations. The following sample trace, `wan_net_delay.trace`, was recorded between one machine at GMD FOKUS and one machine at the Computer Science Department of the University of Massachusetts at Amherst. The trace was done on Aug. 25, 1995 around 13:00 GMT using the tool `rtpdump` [Sieckmeyer95]. The probability density function of the variable part of the observed delays (i.e., delays minus minimum delay) is depicted in Figure 6-15.

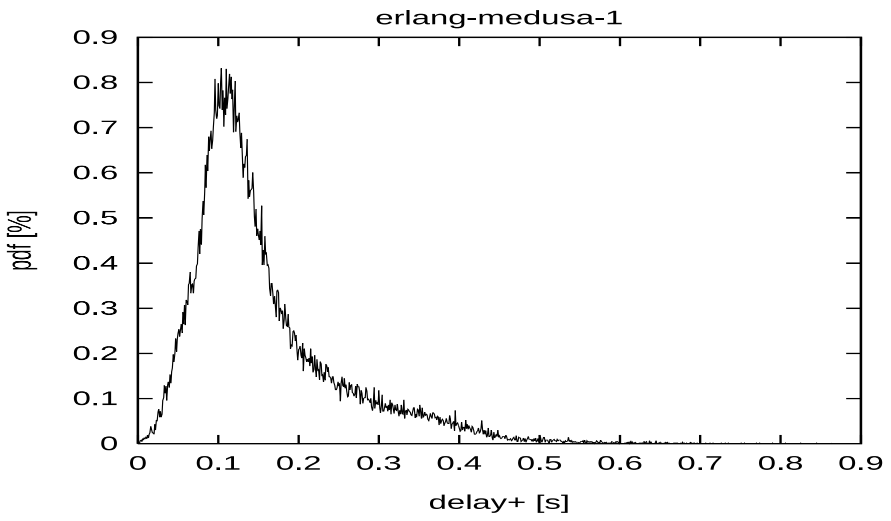

***Figure 6-15*** **A measured delay distribution**

For the easy use of the simulations, the observed variable parts of delays (delays+) are scaled in the range of [0, 1]. For example, the mean delay+ for the above-mentioned `wan_net_delay.trace` is 0.2018.

### 6.4.2 Calibration and validation of simulation models

As a step to further verify the validity of the simulation models used in the above sections, the results of a trace-oriented simulation and a corresponding simulation based on mathematical distributions are compared. The following parameters are used for a real-time process in both simulations:

*compTime* 10, *Period* 20, *Offset* 100, *JobCnt* 2, *Buffersize* 100, *JobOffset* 0, *JobLoss* 0, *JobArrival* 0 / 20 / 0, *netDelay* 30, *playbackDelay+* 0 ...... 800 (step 5).

The differences lie in *JobSize*, *JobDelay* and *NetDelay*. For the theoretical simulations, the following parameters are used:

*JobSize 50%(2/5/0) | 75%(2/7.5/0) | 100%(2/10/0), JobDelay (2/50/0 + 0/50/0).*

That is, the actual computation capacity needs are respectively 50%, 75% and 100% of the reserved computation capacity. The computation sizes of jobs are assumed to be exponentially distributed. The *JobDelay*, which is the variable part of the network delay, is composed of an exponential distribution and a constant.

For the trace-oriented simulations, the following parameters are used:

*JobSize 50%(5/mpeg_frame.trace/37) | 75%(5/mpeg_frame.trace/55.5) | 100%(5/mpeg_frame.trace/74), JobDelay 5/net.trace/495.5.*

That is, the computation sizes and the variable part of delays are taken from the trace files instead of being mathematically generated. Note that the scaling coefficients have been so chosen that the means of the trace data series are equal to their theoretical distribution counterparts. For example, for 75% reservation grade, *mpeg_trace_mean * scaling_coefficient = 0.135 * 55.5 = 7.5*. And for the variable network delay, *net_trace_mean * scaling_coefficient = 0.2018 * 495.5 = 100.*

The results of the theoretical simulations and the trace-oriented simulations are compared in Figure 6-16, Figure 6-17 and Figure 6-18. The three figures correspond to different reservation grades, while the success rates of the trace-oriented simulation and the theoretical simulation for both RELAY and CANCEL timing handling are compared in each figure. It can be seen clearly that the results of the trace-oriented simulation and the theoretical simulation match quite well. Small differences can be found between the corresponding curves, but the general “trends” of the curves agree with each other well. This has further ascertain our simulation models to some extent. And, the other way round, the results from the theoretical simulations can indeed be believed to be realistic and usable. Some numerical examples from the figures are listed in Table 6-8.

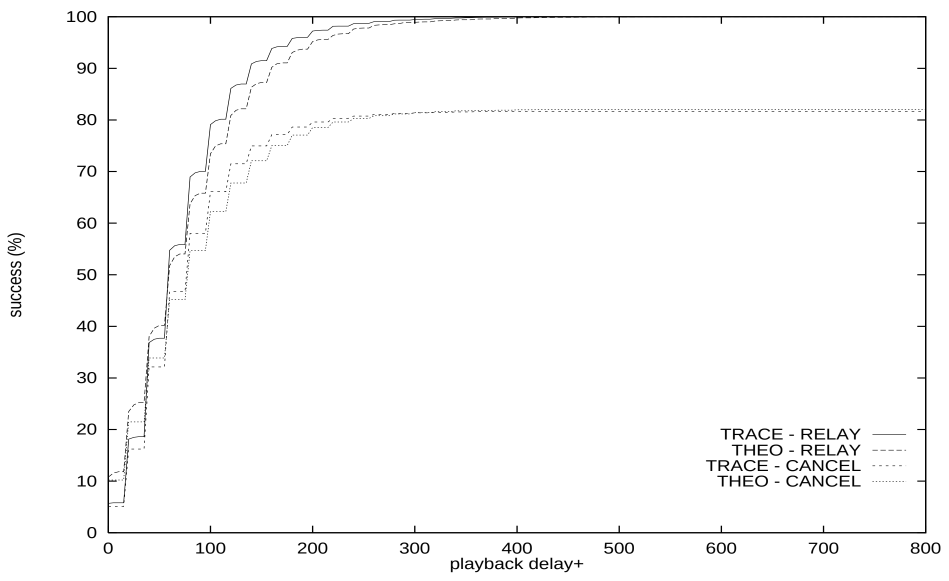

***Figure 6-16*** **Comparison of theoretical and trace-oriented simulations (50% reservation grade)**

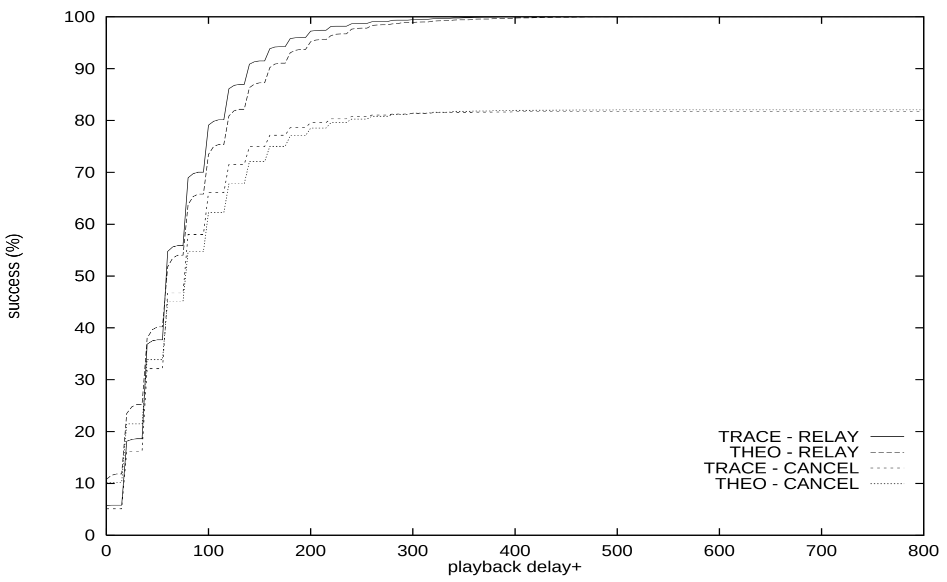

***Figure 6-17*** **Comparison of theoretical and trace-oriented simulations (75% reservation grade)**

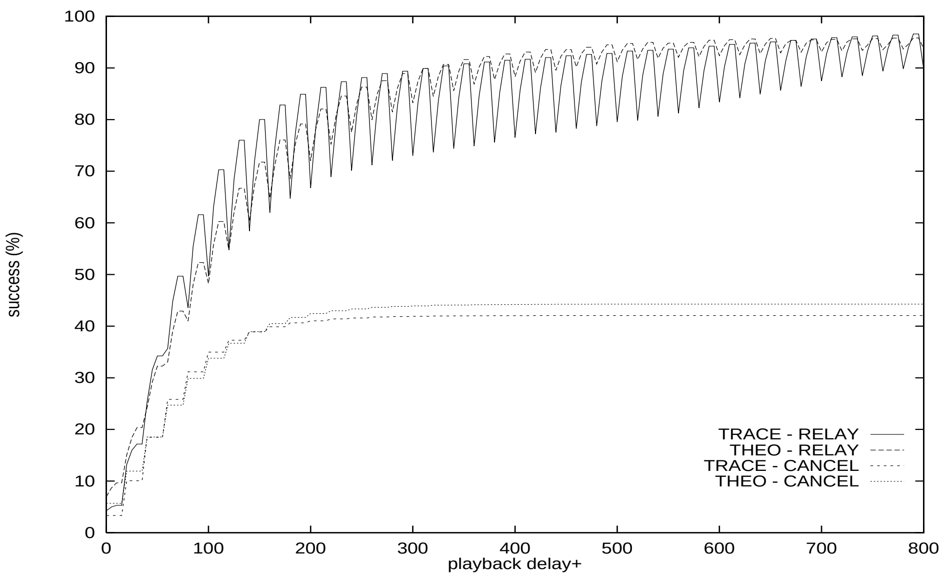

***Figure 6-18*** **Comparison of theoretical and trace-oriented simulations (100% reservation grade)**

***Table 6-8*** **Comparison of theoretical and trace-oriented simulations**

| Success rate in % | RELAY (TRACE / THEO) | | | CANCEL (TRACE / THEO) | | |
|---|---|---|---|---|---|---|
| Playback delay+ | Reservation grade | | | Reservation grade | | |
| | 50% | 75% | 100% | 50% | 75% | 100% |
| 10 | 5.8 / 11.8 | 5.7 / 10.8 | 5.3 / 9.7 | 5.1 / 10.2 | 4.3 / 7.6 | 3.3 / 5.7 |
| 30 | 18.6 / 25.2 | 18.2 / 23.1 | 17.2 / 20.3 | 16.2 / 21.5 | 12.9 / 15.8 | 10.1 / 11.9 |
| 50 | 37.7 / 40.2 | 36.8 / 37.0 | 34.2 / 32.2 | 32.1 / 33.9 | 24.4 / 24.7 | 18.5 / 18.5 |
| 75 | 55.9 / 54.0 | 54.3 / 49.5 | 49.7 / 42.9 | 46.7 / 45.2 | 34.4 / 32.8 | 25.8 / 24.7 |
| 100 | 79.1 / 73.4 | 72.7 / 62.6 | 49.7 / 48.2 | 66.1 / 62.2 | 47.3 / 45.1 | 35.0 / 33.8 |
| 150 | 91.5 / 87.3 | 89.2 / 82.9 | 80.0 / 71.8 | 75.0 / 72.1 | 53.0 / 52.1 | 39.0 / 38.9 |
| 200 | 97.2 / 95.2 | 93.4 / 89.3 | 66.7 / 72.0 | 79.6 / 78.5 | 55.9 / 56.7 | 41.0 / 42.4 |
| 250 | 98.7 / 97.8 | 97.7 / 96.0 | 88.1 / 86.3 | 80.7 / 80.3 | 56.7 / 57.9 | 41.6 / 43.3 |
| 300 | 99.5 / 98.9 | 97.4 / 97.6 | 73.0 / 83.2 | 81.4 / 81.4 | 57.1 / 58.7 | 41.9 / 43.9 |
| 350 | 99.8 / 99.4 | 99.5 / 99.1 | 90.8 / 91.6 | 81.6 / 81.8 | 57.2 / 58.9 | 42.0 / 44.1 |
| 400 | 99.9 / 99.8 | 99.5 / 99.4 | 76.5 / 88.3 | 81.7 / 81.9 | 57.3 / 59.1 | 42.0 / 44.2 |
| 450 | 99.9+ / 99.9 | 99.9 / 99.8 | 92.3 / 93.5 | 81.7 / 82.0 | 57.3 / 59.1 | 42.0 / 44.2 |
| 500 | 99.9+ / 99.9 | 99.9 / 99.9 | 79.5 / 91.2 | 81.7 / 82.0 | 57.3 / 59.2 | 42.0 / 44.3 |
| 550 | 99.9+ / 99.9+ | 99.9+ / 99.9+ | 93.6 / 94.8 | 81.7 / 82.1 | 57.3 / 59.2 | 42.0 / 44.3 |
| 600 | 99.9+ / 99.9+ | 99.9+ / 99.9+ | 83.4 / 92.4 | 81.7 / 82.1 | 57.3 / 59.2 | 42.0 / 44.3 |
| 650 | 99.9+ / 99.9+ | 99.9+ / 99.9+ | 95.0 / 95.7 | 81.7 / 82.1 | 57.3 / 59.2 | 42.0 / 44.3 |
| 700 | 99.9+ / 99.9+ | 99.9+ / 99.9+ | 87.5 / 93.1 | 81.7 / 82.1 | 57.3 / 59.2 | 42.0 / 44.3 |
| 750 | 99.9+ / 99.9+ | 99.9+ / 99.9+ | 96.2 / 95.7 | 81.7 / 82.1 | 57.3 / 59.2 | 42.0 / 44.3 |
| 800 | 99.9+ / 99.9+ | 99.9+ / 99.9+ | 89.7 / 93.8 | 81.7 / 82.1 | 57.3 / 59.2 | 42.0 / 44.3 |

Some simulations have also been conducted to use both trace data and mathematical distributions together. For example, Figure 6-19 depicts the results of a simulation by using *mpeg_frame.trace* and by assuming constant network delay. Other simulations deal with the case with smaller processing variations. For example, Figure 6-20 and Figure 6-21 show the results of the trace-oriented simulations for both RELAY and CANCEL handling by using *mpeg_gop.trace*. Generally, these results have further ascertain a conclusion drawn in the previous sections: the RELAY timing-overflow handling method usually achieves a better results than the CANCEL method.

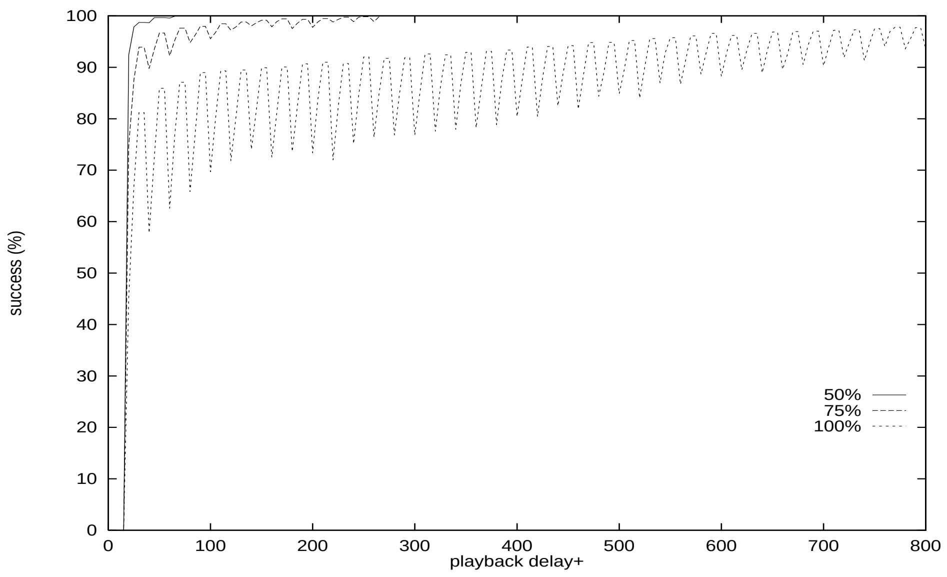

***Figure 6-19*** **A case with real processing cost and ideal network**

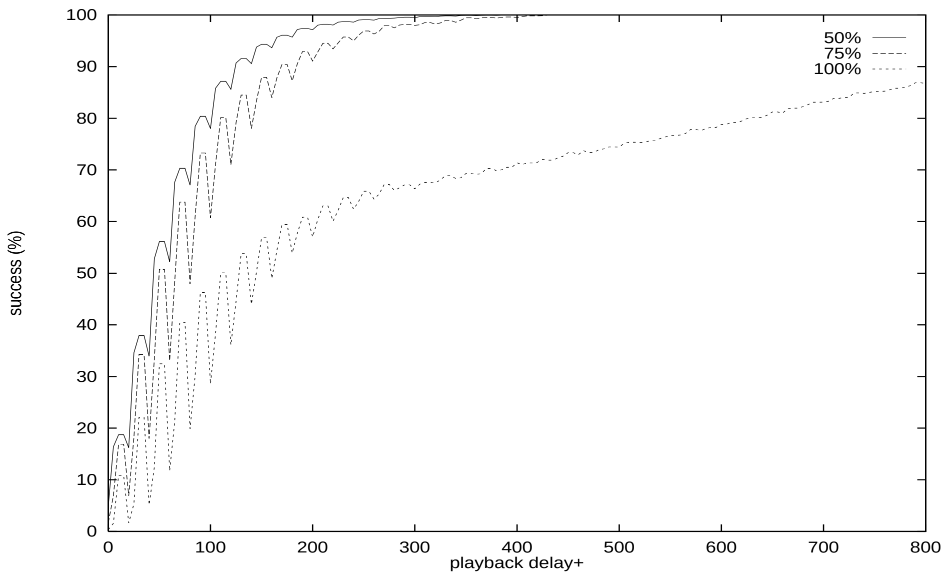

***Figure 6-20*** **RELAY handling of less variant processing cost and real delay**

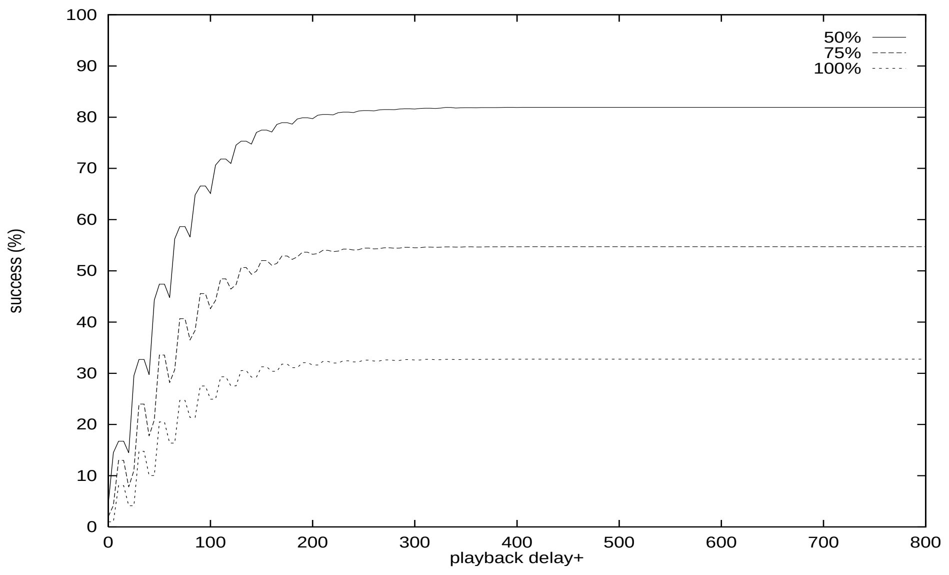

***Figure 6-21*** **CANCEL handling of less variant processing cost and real delay**

***Table 6-9*** **Numerical examples of other trace-oriented simulations**

| Success rate in % | Figure 6-19 | | | Figure 6-20 | | | Figure 6-21 | | |
|---|---|---|---|---|---|---|---|---|---|
| Playback delay+ | Reservation grade | | | Reservation grade | | | Reservation grade | | |
| | 50% | 75% | 100% | 50% | 75% | 100% | 50% | 75% | 100% |
| 10 | 0.0 | 0.0 | 0.0 | 18.7 | 16.9 | 10.9 | 16.7 | 13.0 | 8.1 |
| 20 | 92.4 | 74.7 | 45.2 | 16.2 | 6.9 | 1.7 | 14.5 | 7.8 | 4.1 |
| 30 | 98.7 | 93.9 | 81.2 | 37.9 | 34.3 | 22.1 | 32.7 | 24.0 | 14.8 |
| 40 | 98.6 | 89.8 | 57.8 | 33.9 | 18.0 | 5.1 | 29.7 | 17.8 | 10.0 |
| 50 | 99.7 | 96.7 | 85.9 | 56.1 | 50.7 | 32.5 | 47.4 | 33.6 | 20.5 |
| 75 | 99.9+ | 97.6 | 87.1 | 70.3 | 63.8 | 40.5 | 58.6 | 40.7 | 24.7 |
| 100 | 99.9+ | 95.6 | 69.7 | 78.0 | 60.7 | 28.6 | 65.1 | 42.6 | 24.9 |
| 125 | 99.9+ | 97.9 | 80.6 | 90.7 | 79.2 | 44.6 | 74.6 | 47.3 | 27.5 |
| 150 | 99.9+ | 99.2 | 89.9 | 94.3 | 87.9 | 56.9 | 77.5 | 52.0 | 31.3 |
| 200 | 99.9+ | 97.8 | 73.3 | 97.1 | 91.1 | 57.0 | 79.7 | 53.2 | 31.6 |
| 250 | 99.9+ | 99.8 | 92.0 | 99.1 | 96.9 | 65.9 | 81.3 | 54.4 | 32.6 |
| 300 | 99.9+ | 99.9+ | 76.8 | 99.5 | 98.0 | 66.4 | 81.6 | 54.5 | 32.6 |
| 350 | 99.9+ | 99.9+ | 92.9 | 99.9 | 99.5 | 69.3 | 81.8 | 54.7 | 32.7 |
| 400 | 99.9+ | 99.9+ | 80.4 | 99.9+ | 99.5 | 71.4 | 81.9 | 54.7 | 32.7 |

It is interesting to note that the success rates do not increase linearly with the increase of the playback delay in many of the curves, especially in the areas of small playback delays. The reason lies in the periodic process model and the fact that a process begins to work on a job if the job has not yet missed its deadline. Yet, the job might in fact have missed its playback deadline after it is processed completely. Such a processing is of course a waste of the processing power but it is unavoidable given the fact that a process does not know, beforehand, how long it will take to process the job and when the CPU is available to it during the next period. A small increase of the playback delay at some points might cause more such wastes. The complicated interactions of these several factors have caused the saw-toothed curves in the areas of smaller playback delays. With the increase of the playback delay, the effect is gradually smoothed out. The general trend is then higher success rates with longer playback delays.

## 6.5 Summary of the chapter

The work on simulation aims to evaluate the soft real-time scheduling and handling methods not only qualitatively but also quantitatively.

The construction and realization of our simulation models as well as their parameterizations are based on extensive analyses of the practical application scenarios and current multimedia software packages. In addition to the components reflecting soft real-time scheduling and handling methods, our simulator has also reflected some of the characteristics of continuous media applications and the networking environments. The simulator is composed of several functional components with diverse parameter setting possibilities. It is thus possible to simulate complicated process sets and integrated situations.

The key features of the continuous media applications are embodied in the simulation in the form of play-back scenarios. The main index in the simulation is the rate of successful playback reflecting the quality of the continuous media applications.

The first set of simulations has shown that soft real-time support is necessary and better. Without soft real-time support, it is shown that the quality of the multimedia processing can not be guaranteed to be satisfactory under disturbance or heavy load. As a result of real-time scheduling and timing enforcement, more concurrent multimedia applications can be supported satisfactorily in a soft real-time environment than in a time-sharing environment. A further scenario shows that soft real-time can ensure the timing feature of A/V processes even in a competitive situation. The timing specification of each process is fulfilled without sacrificing the timing reservations of other processes. Also, soft real-time is shown to promote regularity and synchronism: it can make sure that an A/V process can take its share of execution and thus make progress regularly according to its timing reservation.

Through simulations, it becomes apparent that timing monitoring and enforcement is necessary for general and dynamic systems. A scheduling system like SIM-SCHEDULE can monitor the execution of the system and take enforcement measures in case of need. In this way, the reservations and guarantees of timing characteristics of the whole system can be achieved in a predictive and stable manner. Simple cooperative scheduling is only usable in a static, constrained and stable system environment where transient and slight overloads can be more or less compensated between processes silently.

Numerous other simulations have been conducted to compare the influences of the different soft real-time handling methods for timing-overflow. The RELAY and RELAY-TRUNC methods have been found out to achieve better success rates than the CANCEL method as long as the reservations are not made too optimistically. The RELAY method is additionally easy to use and free of the complications caused by the truncations of processing.

In addition to the “theoretical” simulations parameterized by some mathematical distributions, some simulations have also been conducted which are based on the measurements from the real-world scenarios. The results of the theoretical simulations and the trace-oriented simulations have been shown to match well. Our simulation models and their constructions are thus further validated and the usability of the results from the theoretical simulations is also further founded.

In short, the simulations have ascertained the feasibility of our soft real-time framework. They have provided some quantitative evaluations of the soft real-time scheduling and handling methods. Some of the results have been used in choosing algorithms and their parameters in our prototype implementations.

# Chapter 7

# Experimental implementations and measurements

The simulations of the last chapter have evaluated our proposed soft real-time scheduling and handling methods qualitatively and quantitatively. The methods have been compared under different system parameter settings. Those results are very helpful in deciding the applicability of different methods to different environments. In this chapter, our work on experimental implementations and measurements of the soft real-time scheduling and handling methods are reported. These implementations and measurements should further verify the feasibility of our approach and algorithms. The practical experiences will also be very enlightening for more practical work in this direction.

## 7.1 Experimental implementation environment

The experimental implementations and measurements have been conducted on a "TMS320C40 Parallel Processor Development System (PPDS)" produced by the Texas Instruments [TI91]. The choice of the PPDS system was not only due to its technical features described below but also due to the needs of other related project activities [Fan94b, Schatzmayr94, Selent96].

### 7.1.1 Hardware platform

The main hardware platform of the PPDS is an external evaluation board with four TMS320C40 ('C40) processors. The evaluation board can be accessed by a host computer on which the development is aided by a set of development tools.

Texas Instruments' TMS320C4x generation floating-point processors are designed specifically to meet the needs of parallel processing and other real-time embedded applications. Devices in the TMS320C4x generation incorporate on-chip hardware to facilitate high-speed interprocessor communication and concurrent I/O without degrading CPU performances.

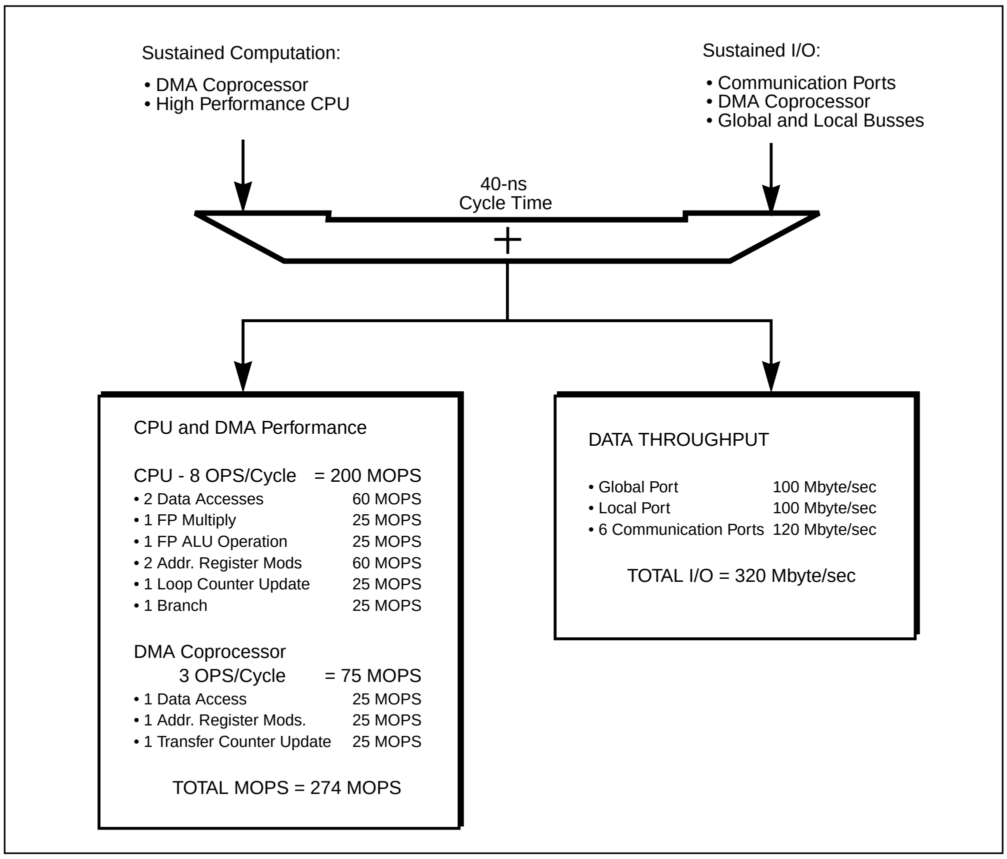
Sustained Computation:
• DMA Coprocessor
• High Performance CPU
Sustained I/O:
• Communication Ports
• DMA Coprocessor
• Global and Local Busses
40-ns
Cycle Time
CPU and DMA Performance
CPU - 8 OPS/Cycle = 200 MOPS
• 2 Data Accesses 60 MOPS
• 1 FP Multiply 25 MOPS
• 1 FP ALU Operation 25 MOPS
• 2 Addr. Register Mods 60 MOPS
• 1 Loop Counter Update 25 MOPS
• 1 Branch 25 MOPS
DMA Coprocessor
3 OPS/Cycle = 75 MOPS
• 1 Data Access 25 MOPS
• 1 Addr. Register Mods. 25 MOPS
• 1 Transfer Counter Update 25 MOPS
TOTAL MOPS = 274 MOPS
DATA THROUGHPUT
• Global Port 100 Mbyte/sec
• Local Port 100 Mbyte/sec
• 6 Communication Ports 120 Mbyte/sec
TOTAL I/O = 320 Mbyte/sec

***Figure 7-1*** **Summary of TMS320C40 performance data**

The main features of the TMS320C40 processor are:

- High-performance DSP CPU capable of 275 MOPS and 320 Mbytes/sec (with reference to a 40 ns CPU cycle time);
- On-chip program cache and dual-access/single-cycle RAM for increased memory access performance;
- Six communication ports for high-speed interprocessor communication;

- Six-channel DMA coprocessor for concurrent I/O and CPU operation, thereby maximizing sustained CPU performance by alleviating the CPU of burdensome I/O;
- Two identical data and address buses supporting shared memory systems and high data rate, single-cycle transfers;
- On-chip analysis module supporting efficient parallel processing debug;
- Separate internal program, data and DMA coprocessor buses for support of massive concurrent I/O of program and data throughput, thereby maximizing sustained CPU performance.

The total performance data is summed up in Figure 7-1. In addition, the 'C40 processor uses a "Load-and-Store" architecture which is comparable to a RISC structure. It has a relative large register file with 32 general-purpose registers and 8 special-purpose registers.

In PPDS, the host-independent evaluation board contains four 'C40 processors. Each 'C40 is connected to every other 'C40 via their communication ports. For our implementation, the 'C40 processor was clocked at 16 MHz. Each processor posses 5 memory areas: an internal RAM of 8 kbytes; a internal periphery area for memory mapping of I/O devices; a local RAM of 64 kbtes, a global RAM of 128 kbytes and a ROM area for boot loader etc. The development of software is conducted on a host computer and then downloaded to the external evaluation board through an interface provided by a parallel processing emulator (XDS510), which also makes parallel debugging and measurement possible.

### 7.1.2 Procedure of software development

As is shown in Figure 7-2, the development of software for the PPDS involves the use of an ANSI-C compiler, an assembler, an archiver, a linker, two libraries as well as a simulator and a XDS emulator.

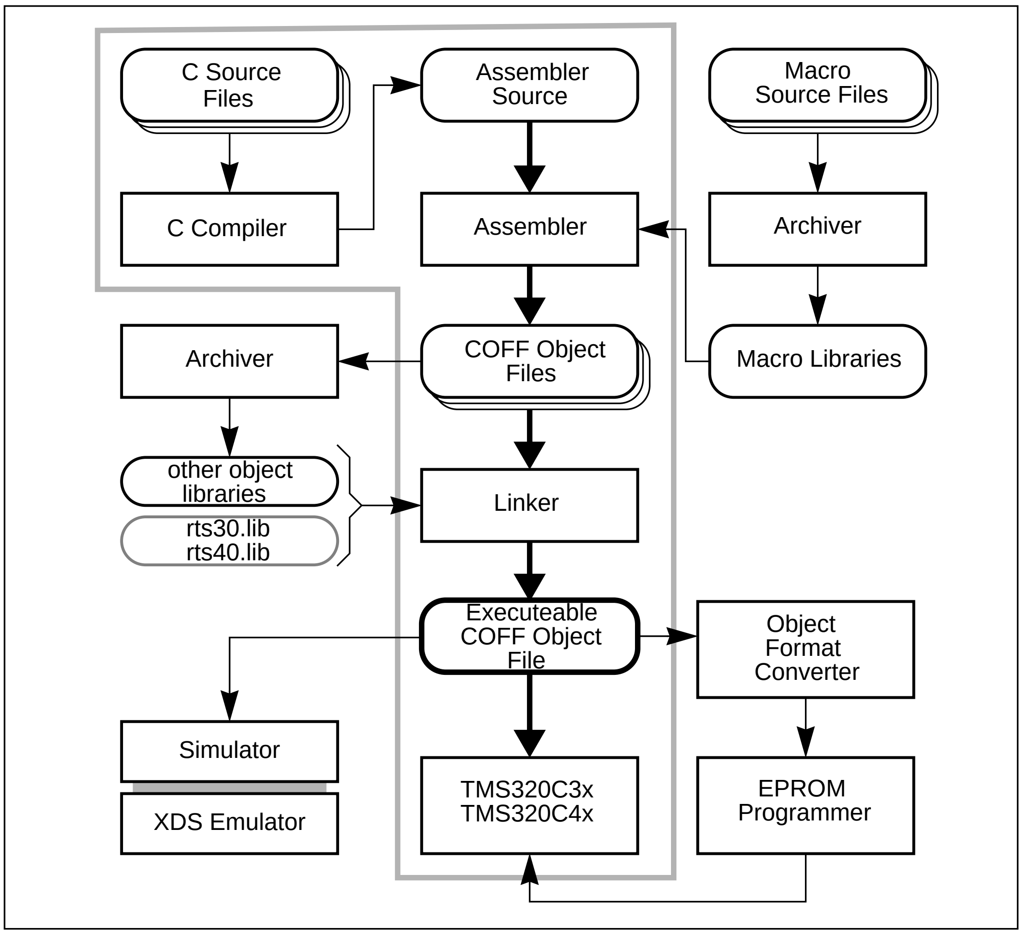

***Figure 7-2*** **Procedure of software development for PPDS**

## 7.2 Overview of the implementation

The original goal of our experimental implementation in the PPDS environment was to implement a run-time supporting system for a communication subsystem which may be, for instance, implemented on a front-end subsystem. The construction of the run-time supporting system has followed the micro-monolithic approach as described in *Section 5.5.2 A micro-monolithic-kernel approach.*

### 7.2.1 System components

The main components of the run-time supporting system are depicted in Figure 7-

3. As can be seen from the figure, the run-time supporting system provides many of the main functions of a normal operating system. Our goal was not to implement a complete OS. The hardware environment had also prevented us from doing so. For example, the ‘C40 processor does not differentiate between supervisor mode and user mode. This makes it impossible to have a strict boundary between kernel space and user space as in the normal sense. As another example, the ‘C40 processor does not provide a mechanism for prioritizing and masking interrupts either. However, all these restrictions do not prevent us from implementing a relatively complete run-tim supporting system. It is thus still possible for us to gain useful insights into the realization of a soft real-time supporting system.

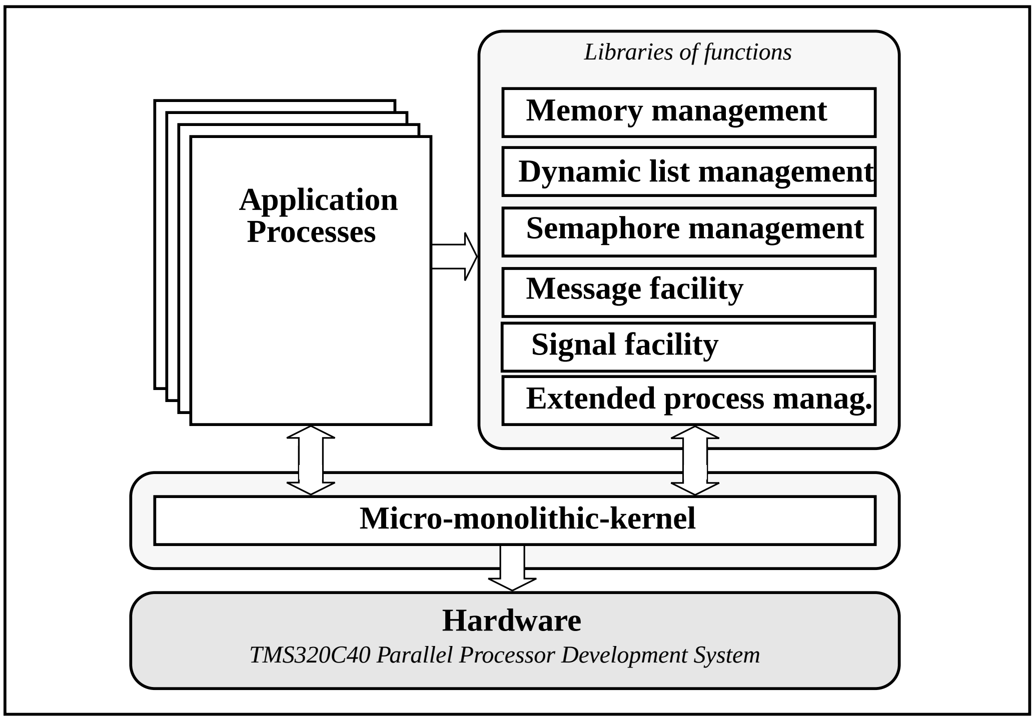

***Figure 7-3*** **System components of the run-time supporting system**

The components provide functionality similar to those provided by their counterparts in a normal OS, although sometimes in reduced forms:

- The memory management component manages dynamically the free memory blocks and the memory blocks currently in use. It takes into account the fact that the PPDS system board possesses several different memory areas of different types.

- The dynamic lists management provides a set of functions for the manipulation of dynamic data structure in the general form of lists.
- The semaphore management component makes it possible for processes to guard their critical section by using semaphores. A simple form of priority inheritance has been implemented to alleviate priority inversion problem.
- The signal facility can be used by the processes to inform and get informed about asynchronous events. One can `poll` or `wait` for an event, which will eventually be `signal`led. A general form of asynchronous signal handler is, however, not provided.
- The message facility makes it possible for the processes to exchange messages between established `port`s. Only asynchronous message passing is supported.
- The kernel and extended process management provide a set of facilities for managing and scheduling real-time, virtual real-time and non-real-time processes. Here lies the main emphasis of our implementation. More details will be given in the following.

### 7.2.2 Sample programming interface of usage

The CO-SCHEDULE and SIM-SCHEDULE soft real-time scheduling schemes proposed in Section 5.2 are implemented in the experimental system. For timing-overflow handling in SIM-SCHEDULE, both RELAY and RELAY-TRUNC methods proposed in Section 5.3 have been implemented and can be used for different needs. Programming interfaces for periodic real-time processes and virtual periodic real-time processes have been provided in such a way that the user of the processes only need to provide task body and specify real-time constraints such as start time, period, execution-time. The real arrangement of the periodic instantiations of the task body is then done by the system automatically in a similar scheme as Figure 4-2.

In the implementation, a process or a set of processes should be created before the process or the set of the processes as a whole can be started. The motivation of separating creation and starting of process(es) is that these are independent steps. The creation of process(es) may be quite time-consuming and it is done in a virtual periodic system process or a system process of normal priority class 1. To start a process or a set of processes, the admission control should be done. For the case of a set of processes, the admission control might have been done previously or even off-line by other tools. The implementation details are described in the Section 7.3 followed.

Together with the considerations on admission control and process adaptation, several sets of programming interfaces for process management have been implemented for the experimental system. A simple set of programming interfaces is presented in this subsection schematically for the case of separate and simple admission control of single processes.

The programming interface for creating a semi-imperative periodic process looks like the following:

```
ProcID = CreateProc_PERIODIC (env, unit_prog, end_prog,
                              period, unit_comp_time, tv_excep)
```

where, *env* is a structure containing environmental settings such as stack size and initial arguments, *unit_prog* is the entry point of a procedure which should be executed in each period, *end_prog* is the entry point of a procedure which should be executed when terminating the process, *period* and *unit_comp_time* are the intended period and computation time per period of the intended virtual periodic process, *tv_excep* is the exception handler which should be called by the OS when a timing violation arises. The simplest form of *tv_excep* is the termination of the whole process. Note that *end_prog* and *tv_excep* can be set to NULL to indicate that no corresponding actions should be taken.

To create a virtual periodic process, the following interface can be used:

```
ProcID = CreateProc_VIRTUAL (env, body_prog, end_prog,
                             period, unit_comp_time)
```

where, *body_prog* is the entry point of a procedure which should be executed in the virtual periodic frame, other parameters have the similar meanings as in *CreateProc_PERIODIC*.

After being created, a real-time process should be explicitly started. As will be detailed below, it is at the time of trying to start a process that the system makes an admission control. An already successfully created periodic process or virtual periodic process can be started using the following:

```
success = StartProc_RT(start_mode, ProcID, start_time, end_time)
```

where, `start_time` and `end_time` are the intended start time and end time of the intended (virtual) periodic process. Different `start_mode`'s can be used for different requirements concerning guarantee and timing wishes. It will be further detailed in *Section 7.3.2 Simple admission control*.

Correspondingly, `CreateProc_NORMAL()` and `StartProc_NORMAL()` are provided to create and start a normal process which takes parameters such as normal priority class, quantum, nice grade, etc. In the experimental system, all the processes are allowed to mask some small parts of its process body as non-preemptable with a pair of `FORBID() / PERMIT()`.

## 7.3 Implemented features and implementation trade-offs

In implementing the experimental system, efforts have been made to realize and validate the strategies and algorithms proposed in the previous chapters. Some trade-offs have been made due to the concrete experimental system conditions. Some implemented features and the related implementation trade-offs are presented below with an emphasis on scheduler and scheduling-related functions.

### 7.3.1 Efficient dispatcher

As depicted in Figure 7-4, a process in the experimental system has usually the following life cycle: a process has a state of `ADDED` after being created; after being started, the process is `READY` and can be eventually scheduled to `RUN`; it may change its state into `WAIT` for waiting some resources or events; after running to an end or being killed, the process remains as `ZOMBIE` and will be eventually cleared by a house-keeping system process.

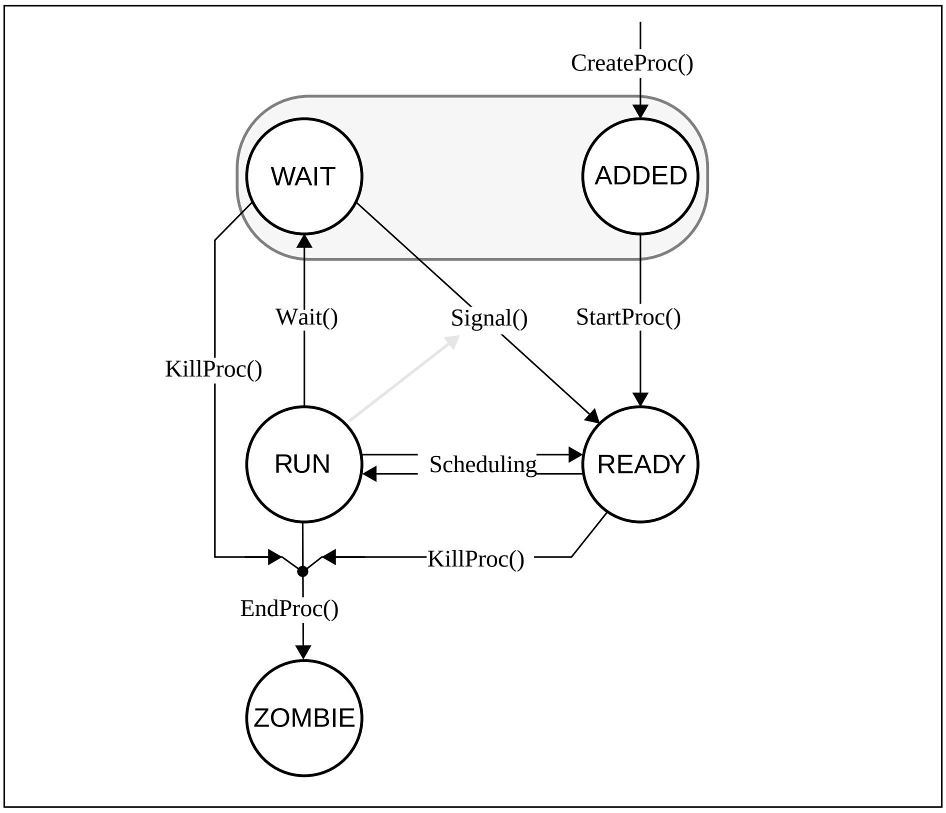

***Figure 7-4*** **Transition of process state**

It is of course very important to have a very low scheduling cost, otherwise the goal of soft real-timeliness can not be achieved. A scheduling scheme such as CO-SCHEDULE or SIM-SCHEDULE can indeed be implemented in a very efficient manner — based in part on a priority-driven dispatcher.

Recall the GRMT results given in Section 4.2 and Section 5.1, the processes under rate-monotonic scheduling are scheduled according to their (static) priorities at run-time. A CO-SCHEDULE scheduler can therefore be derived from a priority-based dispatcher which can be easily implemented efficiently. The high-priority periodic processes are assigned different priorities according to their periods. If necessary, a period-transformation should be done. A SIM-SCHEDULE scheduler must, of course, do the extra work about monitoring and enforcing.

Once again, in contrast to a pure fixed-priority scheduling method, a method based on rate monotonic scheduling has two attributes: (1) it can do admission control in a very simple way by checking the processing capacity required by the processes, especially in the context of soft real-time; (2) the priorities of the processes can be dynamic and will be adjusted when new processes are admitted into the system. Note admission control can be made even for some very complicated scenarios, it is therefore more theoretically sound than a simple priority setting scheme.

Similar to communication bandwidth, the C/T ration for a periodic process indicates the process' share of the CPU bandwidth. Scheduling and enforcement according to GRMT ensures that the processes do not influence each other negatively too much. I.e., each "guaranteed" process is assured of its share of its process bandwidth.

In our scheduler, the mapping of real-time and virtual real-time periodic processes to their priorities are dynamic and are done implicitly by the scheduling subsystem. This is in contrast to a pure priority system where priority setting has to be done explicitly and it is difficult to do reasonable admission control and priority adaptation.

As such, the scheduling of high-priority periodic set is in the end conducted on the base of priority setting. The normal priority processes are scheduled according to a weighted round-robin method.

### 7.3.2 Simple admission control

In contrast to hard-real-time systems, admission control in a soft real-time system is simplified to a large extent. For more complicated situations, the GRMT theories concerning blocking, synchronization etc. can be used to analyze the schedulability of a set of related processes in order to make a precise admission decision. For most simple cases, simple admission tests are good enough to maintain a high success degree. These simple tests generally do not need to account for the intricate dependences among processes. Small factors involving blocking and synchronization can be implicitly smoothed out by the soft real-time features. In the most situations simulated in Chapter 6, for example, a very simple admission control is generally used to simply make sure that the whole capacity to be reserved by the processes is less than the utilization bound of the classic rate-monotonic theory. The simulation results indicate that a high success rate is still achieved although the processes are actually also influenced by factors such as suspension, delayed-ready, etc.

A very simple admission control for separate admission of single processes can be implemented by simply checking the whole capacity already reserved by the existing processes and the capacity needed by the incoming process(es). Note that, in contrast to a pure fixed-priority scheduler, the priorities of the existing real-time processes will have to be adjusted when new processes are admitted into the system. For example, if a new process comes with period $T_{p1} < T_{new} < T_{p2}$, then its priority has to be $P_{p1} < P_{new} < P_{p2}$. If it happens that $(P_{p1} + 1) = P_{p2}$ then either $P_{p1}$ or $P_{p2}$ has to be adjusted.

A closer look at the `start_mode` for starting a real-time process might help to clarify the idea more clearly.

In `start_mode`, a process can request to be started "*now*" (actually "*as soon as possible*") or "*at a specified time later*". In the implementation, a simple record of "process history" has been maintained in the form of a simple database. The reservation for a future time can thus be safely conducted.

Also in `start_mode`, a user should indicate his "*requirement grade*" to indicate what kind of process he intends to start and what kind of process he would like to accept if the best quality can not be satisfied because of admission control. For the "*requirement grade*", *SureG* and *MaybeG* bits may be set which denote "sure guarantee" and "maybe guarantee" requirements or intentions respectively. The system will also try to satisfy the process in this requirement order. The admission control is done by the on-line Admission Controller of the scheduler. For example, a process A has already been created as a real-time periodic process. It should now be started with *SureG* and *MaybeG* bits set. The system will then check if an admission control for a sure-guarantee can be passed. If yes, then the process will be started as a process with sure-guarantee. If no, the system will then see if an admission control for a maybe-guarantee can be passed. If yes, then the process will be started as a process with maybe-guarantee. Otherwise the process can not be started.

- **Implementation scheme of sure-guarantee and maybe-guarantee**

As is stated, the outcome of the decision of the Admission Controller can be one of the following three cases: a sure-guarantee of the real-time attributes of the new process can be maintained; a maybe-guarantee of the real-time attributes of the new process can be maintained; or only a best-effort maintenance of the real-time attributes of the new process can be done.

Recall again that the runtime behavior of the rate monotonic scheduling is simply dependent on a fixed priority assignment scheme. This factually ensures that the processes with higher priorities are directly more favored than the processes with lower priorities under transient load. Recall also that a capacity bound is usually used for various admission control of GRMT. Let's call the set of the processes with total capacity within a certain bound B the B-bounded processes. The smaller the bound is, the fewer processes will pass admission control. Thus, a smaller bound means, to an extent, a lower degree of the risk of possible timing violation. For $B1<B2\leq1$, the B1-bounded processes get higher priorities than the processes admitted in the scope of (B1, B2). This will factually ensure that the B1-bound processes are directly more favored under a transient load than the processes out of the bound.

With these considerations, the following two simple algorithms are implemented for the admission control of separate processes. In these algorithms, two bounds are used for admission control — a sure-bound and a maybe-bound. Initially, the bounds are set to SureBound = $n(2^{(1/n)}-1)$, MaybeBound = maximum (SureBound, 0.88). The sure-bound is taken from Theorem 4-1 for guaranteeing the schedulability of a simple periodic task set with arbitrary phasings. For maybe-bound, the bound is relaxed, in case of need, to 88%, which is the likely schedulability bound for a randomly chosen task set [Lehoczky89].

In line of the above considerations, these two bounds can/should actually be adjusted in a concrete practical environment through experiments and according to experiences.

For the experimental system, the following two algorithms have been used to make admission control for sure-guarantee and maybe-guarantee. Assume a new process with period T and per-period computation time C arrives and it requires a guarantee as strong as possible, the first algorithm tries to place the processes which need sure-guarantee and the processes which need maybe-guarantee in a mixed manner:

```

if (C/T + (the capacities of all existing SureG processes) +
     (the capacities of all existing MaybeG processes with
     period <= T)) <= SureBound
then the new process can be admitted as a SureG process
elsif (C/T + (the capacities of all existing SureG processes) +
     (the capacities of all existing MaybeG processes))
     <= MaybeBound
then the new process can be admitted as a MaybeG process
else the new process can be admitted as a BestEffort process

```

***Figure 7-5*** **Admission control algorithm for mixed placement**

The second algorithm tries to place *MaybeG* processes in a less prioritized position than all *SureG* processes thus make it possible to admit more *SureG* processes into the system in case of need.

```

if (C/T + (the capacities of all existing SureG processes))
     <= SureBound
then {adjust existing MaybeG processes in case of need;
     the new process can be admitted as a SureG process
     }
elsif (C/T + (the capacities of all existing MaybeG processes))
     <= (MaybeBound - SureBound)
then the new process can be admitted as a MaybeG process
else the new process can be admitted as a BestEffort process

```

***Figure 7-6*** **Admission control algorithm for separate placement**

For sake of easy implementation, a *MaybeG* process in consideration is transformed into a virtual periodic process with a virtual period longer than all the *SureG* processes — by giving it a virtual period which is an integral multiple of its original period and activating the integral multiple of its periodic body in this virtual longer period. In this way, its corresponding priority is assigned correspondingly to be lower than all the *SureG* processes so that it is not possible for it to possibly violate the timing constraints of all *SureG* processes.

As mentioned in Chapter 3, the admission control with regard to the schedulability of a complicated set of interdependent processes could be a time-consuming task which need not be conducted on-line. An off-line tester such as Scheduler-1-2-3 could be used to make the schedulability test and the resulted parameter-settings could then be submitted to the running system together. Due to the flexibility and tolerance of the soft real-time systems, the strictness for their admission control can be relaxed. A set of closely related real-time processes, which implement an application together, should be submitted together to the system for a more precise schedulability analysis and, if admitted, corresponding reservations should be done. Groups of processes implementing different applications, though they might be dependent on each other loosely by way of asynchronous communications, might be treated as independent in admission control. In this way, some simple heuristics can be used to simplify admission control for different environments. In this regard, the engineering considerations for applying GRMT to hard real-time applications are also partly applicable [Katcher93].

### 7.3.3 Scheduler as a predictor and an enforcer

Although much work was done in predicting or measuring the processing capacity requirements of a real-time program [Colnaric90, Wittig94], it is always difficult to estimate the exact processing requirements of a program. There are several reasons to this. One is that the runtime behavior of a program is quite dependent of the concrete environments.

To alleviate the problem and to assist users, our idea is to make the scheduler function both as a timing enforcer and a timing predictor. Our scheduler is a timing enforcer in the sense that the scheduler makes timing control by real-time scheduling. Our scheduler is a timing predictor in the sense that it can facilitate the estimation of processing time usage by providing such timing accounting information to the users.

In other words, the difficulty of estimating computation time can be eased by allowing the application processes a “run and adjust” cycle: a real-time process can be delivered with a preliminary set of timing constraints to run; the system feeds back timing accounting information; the process can be run again with a set of adjusted timing constraints; and so on. Through such a cycle of interaction between application processes and the scheduler, a better “cooperation” can be achieved — a better system effect can be achieved both for system under CO-SCHEDULE and SIM-SCHEDULE.

For sake of simplicity, the current implementation instructs the scheduler either to run in a predictor mode or in an enforcer mode. In the enforcer mode (i.e. normal mode), the scheduler only makes essential timing accounting to run as efficiently as possible. In the predictor mode, the scheduler makes a more detailed timing accounting with regard to periods and timing violations. The accounting results can be retrieved by way of a system call from time to time.

### 7.3.4 Implementation of soft real-time handling

For the execution under SIM-SCHEDULE, both RELAY and RELAY-TRUNC timing overflow handling methods are implemented and can be chosen by the users. The users can optionally provide a timing-violation handler (TV-handler), which will be triggered to execute automatically when a timing overflow occurs. In the TV-handler, the user processes can, for example, account the recent timing violation frequency and take corresponding measures later.

The implementation schemes of RELAY and TV-handler are explained in Figure 7-7 and Figure 7-8. In Figure 7-7, a timing-overflow is detected by the system, the violating process is preempted and not-yet-completed part is relayed to be executed in the following period. As can be seen in Figure 7-8, the execution of the TV-handler is also charged against the timing budget of the process itself. This has the advantage that the timing irregularity of a process will not have negative influences on other processes.

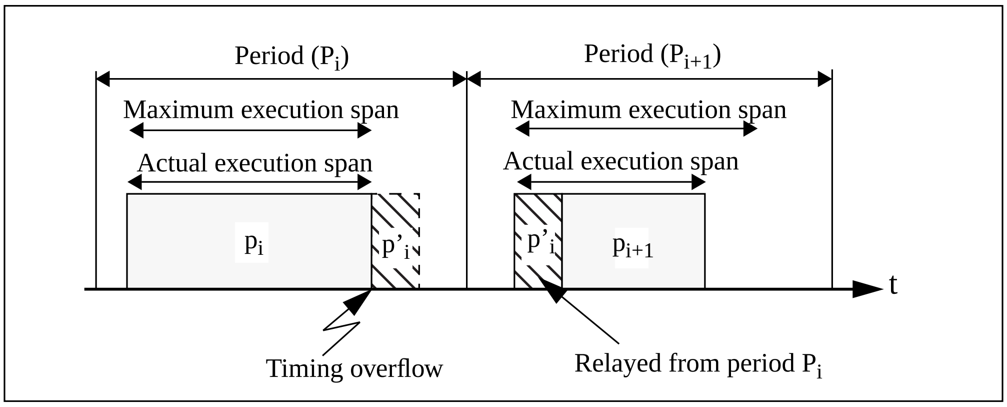

***Figure 7-7*** **Handling with RELAY**

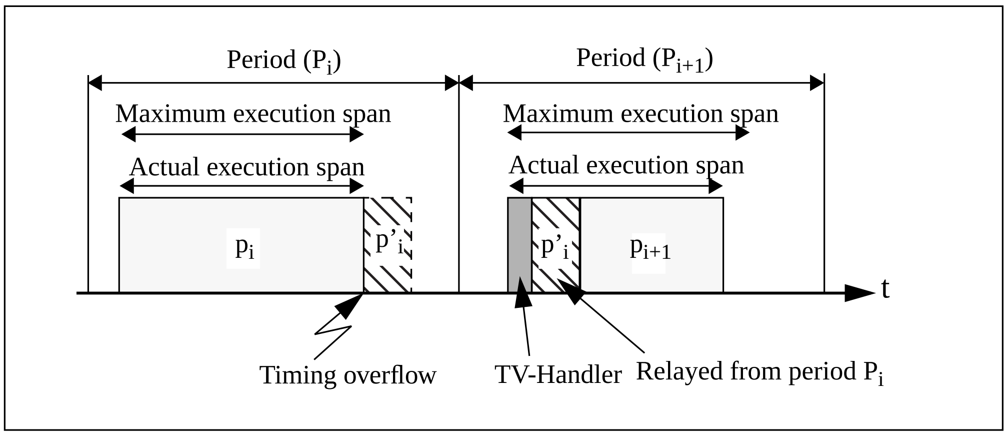

***Figure 7-8*** **Handling with TV-handler and RELAY**

An alternative implementation would be to implement the TV-handler as a separate thread and to let it run concurrently with the new instantiation of the next period. But this would incur much sophistication both in accounting and concurrency control and was therefore not adopted.

### 7.3.5 Low-level scheduling details

As can be seen from above, a real-time process will begin to compete for CPU after it has passed an admission control and got assigned a priority according to the GRMT scheme. The low-level scheduling (dispatching) of the processes is then more or less driven by the priority assignment.

Some details of the low-level scheduling implementations are provided here to make it easy to understand the related measurements to be presented later. More specifically, the processing flow depicted in the following flowcharts (Figure 7-9 and Figure 7-10) has been used to realize the process state transition depicted in Figure 7-4.

The main entry to the dispatcher is the *_RSWITCH* triggered by the periodic scheduling clock interrupt (default to be of 1 ms interval). Basically, the section from *_RSWITCH* to the point numbered 4 is responsible for a possible switch of a process from *RUN* to *READY*. The section from the point 4 to *RETI* (return from interrupt) is responsible for switching a process from *READY* to *RUN*.

The entry point *_WSWITCH* is entered when a process has to transfer into *WAIT* state. This can happen when the process explicitly invokes *WAIT()* or when it has to wait for some resources. The entry point *_NSWITCH* is entered when the process runs to an end or gets aborted. In this case, the process in question is transformed into the state *ZOMBIE*. Both these entries will lead to point 4 since another process will have to be scheduled to run.

System timer processing is necessary to realize the functions such as the automatic arrangement of the periodic process body. Figure 7-10 sketches the flow for system timer processing.

Basic timing accounting and the possible installation of a TV-handler are also parts of the processing flows.

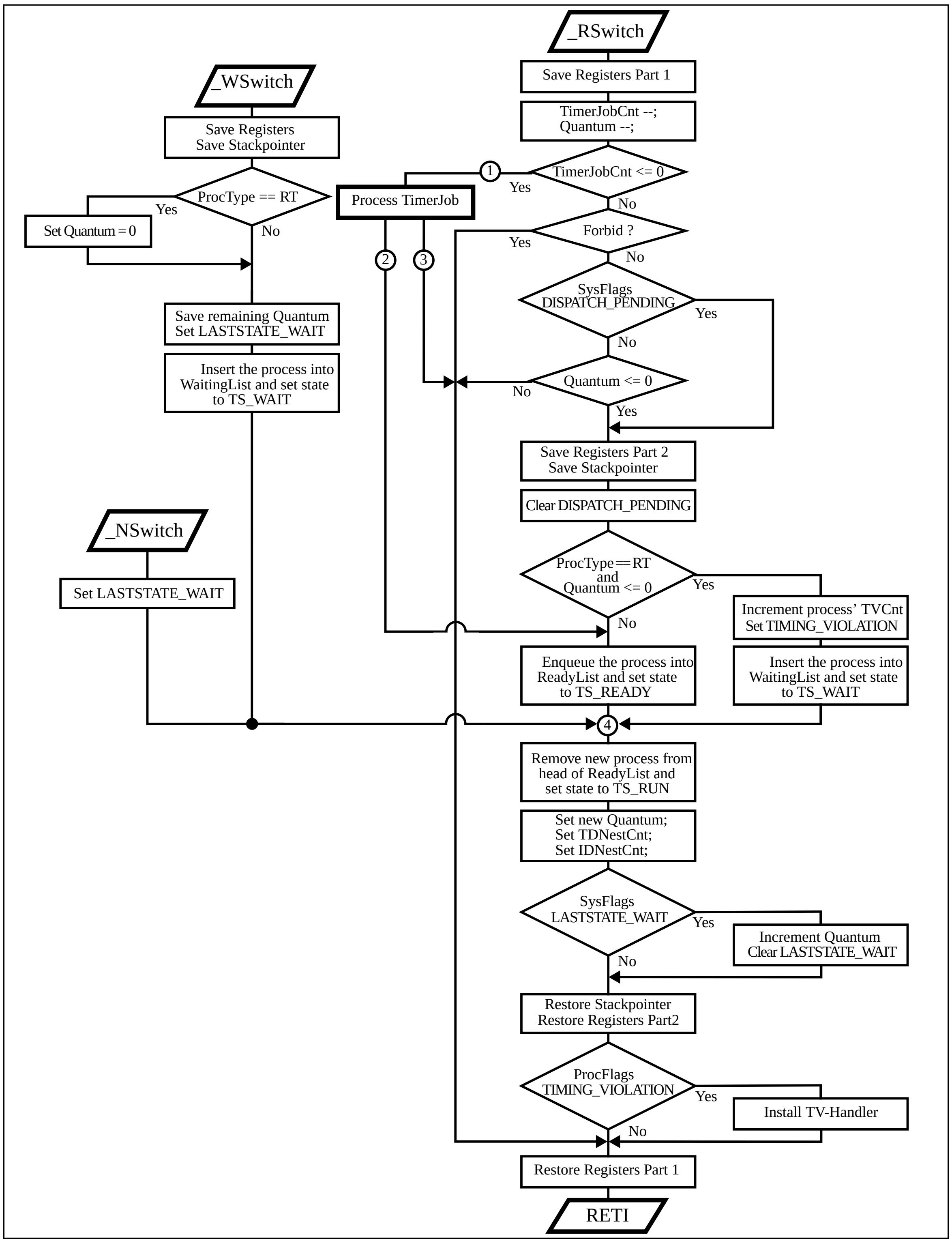

***Figure 7-9*** **Flowchart of low-level process dispatching**

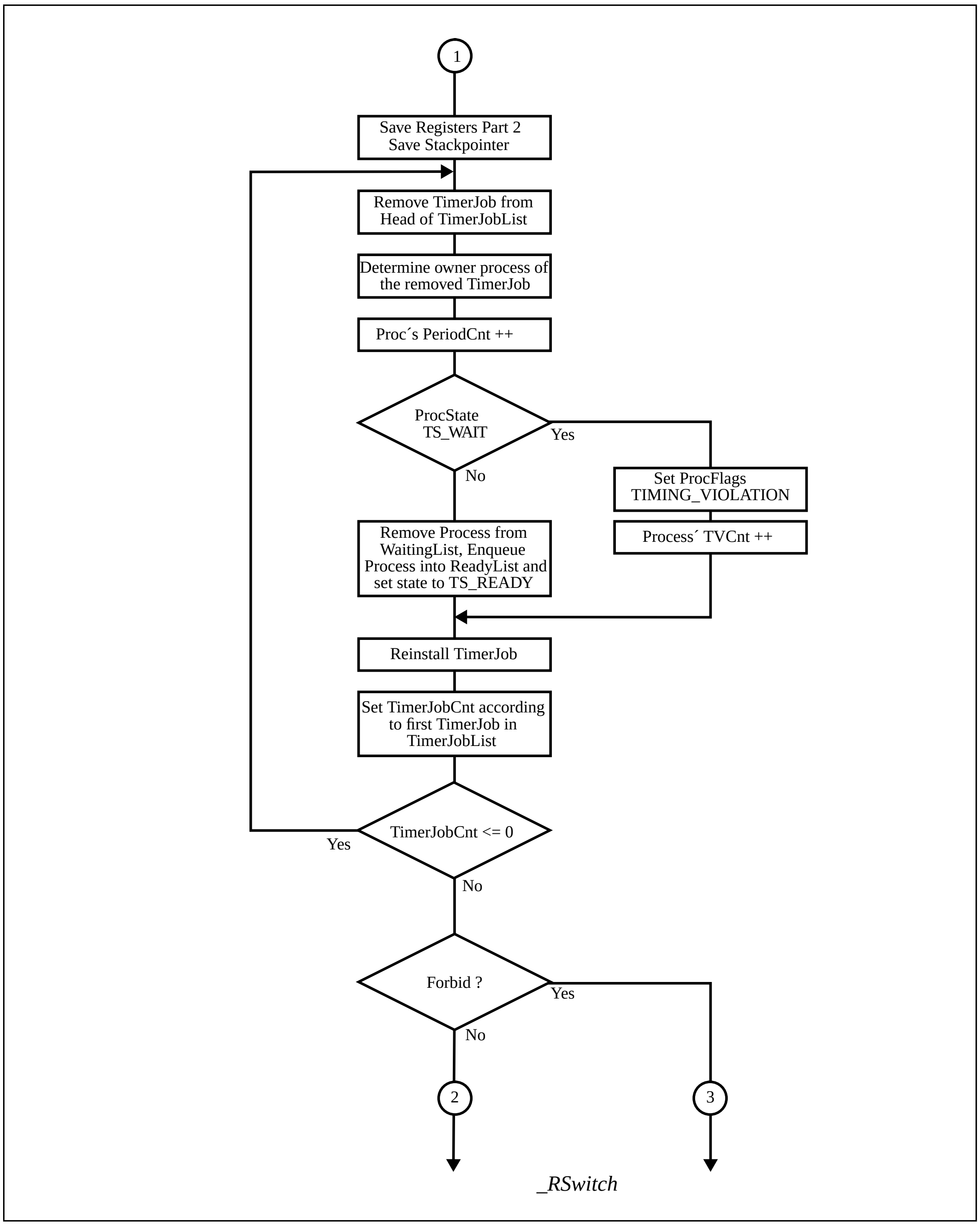
1
Save Registers Part 2
Save Stackpointer
Remove TimerJob from Head of TimerJobList
Determine owner process of the removed TimerJob
Proc´s PeriodCnt ++
ProcState TS_WAIT
Yes
No
Set ProcFlags TIMING_VIOLATION
Process´ TVCnt ++
Remove Process from WaitingList, Enqueue Process into ReadyList and set state to TS_READY
Reinstall TimerJob
Set TimerJobCnt according to first TimerJob in TimerJobList
TimerJobCnt <= 0
Yes
No
Forbid ?
Yes
No
2
3
_RSwitch

*Figure 7-10* **Flowchart of timer-job handling (subfigure of Figure 7-9)**

### 7.3.6 Other features of interest

In the experimental system, most of the system management operations are executed either in the scope of a virtual real-time process or in the scope of a process of normal priority class 1. Examples can be found in creating, starting and clearing processes and other house-keeping operations. Principally, these operations are not as time-critical as the real-time processes which are running. Such an arrangement is helpful to make the main system scheduling flow as light-weighted as possible.

The semaphore management component makes it possible for processes to guard their critical section by using semaphores. A simple form of priority inheritance has been implemented to alleviate priority inversion problem.

The experimental system also provides support for the adaptations of soft real-time processes. This will be described in Chapter 9 in relation with the provision of an adaptive service provision model (see *Section 9.3.4 Adaptation support from OS scheduler*).

## 7.4 Performance measurements and analyses

It is clear that the goal of soft real-timeliness can not be achieved if the proposed models and mechanisms can not be implemented to perform efficiently. Measurements have been made to asses the performances of our experimental implementations. The measurement results have confirmed that it is easy to implement efficient base schedulers incorporating our soft real-time scheduling and handling methods.

### 7.4.1 Measurement setup

The measurements were conducted on a PPDS evaluation board with TMX320C40 processors. The TMX320C40 is a functionally identical pre-version of TMS320C40. The processors were clocked at 16 MHz.

The test programs and the run-time system components, with the exception of the interrupt handlers `_RTClock`, `_NSwitch`, `_WSwitch` and `_RSwitch`, were resident in the local RAM area. For efficiency reasons, the above-mentioned four interrupt handlers were loaded into internal RAM to run.

The global RAM area of the PPDS evaluation board was not used in the measurements so that the synchronization interdependence between processors had no effects on the measurement results. The instruction cache was also deactivated so that a relatively stable and less optimistic results could be gained.

## 7.4.2 Basic scheduling overhead

A scheduling scheme such as SIM-SCHEDULE can indeed be implemented in a very efficient manner, partly due to the reason that its runtime behavior is based on a static-priority-driven dispatcher to a large extent. The following tables give some examples of the basic scheduling overhead of the SIM-SCHEDULE as we have implemented it on the TMS-C'40 processor. The basic overhead includes the overheads for context switch, basic timing monitoring and enforcing etc. For SIM-SCHEDULE, this corresponds to what is depicted in Figure 7-9. But it does not count into its subfigure Figure 7-10 concerning “timer handling” and the part concerning “installing timing-violation handler”, which are dealt with in the next subsection.

The numerical examples listed in all the following tables are the maximum overheads which are not usually needed in a normal execution scenario. Note that the TMS-C'40 is a RISC-like processor with many registers. The scheduling overheads could have been reduced greatly if the operations for register switch had been reduced.

In Table 7-1, some measurements of the basic scheduling costs of SIM-SCHEDULE implemented on PPDS are listed. The measurements are listed both in the unit of instruction cycles and microseconds. In the experimental implementation, the scheduling interval is set to 1 ms, since the time granularity of the multimedia applications are mostly in the range of miniseconds. The low-level scheduler is thus triggered each minisecond or at the event when a process changes its status into ready. If a highest priority real-time process is running within its time quantum, then no process context switch is needed. In this case, the scheduling overhead is quite small. In the case of a context switch, the scheduling overhead is dependent on the current number of ready real-time processes, since the pre-empted process should be put into the right waiting position among all ready real-time processes sorted by their priorities. Note that the scheduling overheads will not increase with the number of non-real-time processes which will not get scheduled as long as there are ready real-time processes.

***Table 7-1*** **Measured maximum basic scheduling overhead**

| Basic scheduling overhead | Within quantum | Number of ready real-time processes | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 5 | 10 | 20 |
| Overhead in cycles | 57 | 243 | 251 | 264 | 290 | 355 | 485 |
| Overhead in μs | 3.56 | 15.19 | 15.69 | 16.50 | 18.13 | 22.19 | 30.31 |

Nowadays, it is common for the microprocessors to clock at over 100 MHz or even 200 MHz. In Table 7-2, the measurements are proportionally mapped onto processors with 120 MHz and 300 MHz clock respectively. The maximum basic scheduling overheads in percentage can be seen to be quite small with regard to a scheduling interval of 1 ms.

***Table 7-2*** **Proportional maximum basic scheduling overhead in percentage**

| System clock frequency | Within quantum | Number of ready real-time processes | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 5 | 10 | 20 |
| 120 MHz | 0.048 | 0.203 | 0.209 | 0.220 | 0.242 | 0.296 | 0.404 |
| 300 MHz | 0.019 | 0.081 | 0.084 | 0.088 | 0.097 | 0.118 | 0.162 |

### 7.4.3 Additional overhead with full real-time support

Under SIM-SCHEDULE, the system provides some direct timing supports for real-time processes. The periodic process bodies are instantiated by the system. Such instantiations involves timer handling which is dependent on the number of expiring system timers. The maximum additional overheads for such timer handling are measured and listed in Table 7-3.

***Table 7-3*** **Measured maximum additional scheduling overhead**

| Additional scheduling overhead | Number of expiring system timers | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 5 | 10 | 20 |
| Overhead in cycles | 101 | 128 | 155 | 209 | 344 | 614 |
| Overhead in µs | 6.31 | 8.00 | 9.69 | 13.06 | 21.50 | 38.37 |

In our implementation, the number of the system timers will not be very large. The timer pools needed by subsystems such as protocol processing are processed either by a virtual real-time process or a system process of normal priority 1.

The instantiation of a periodic process body will be activated at each beginning of the period of a real-time process. A longer period implies thus a smaller overhead. Assume a system runs a set of real-time processes with periods of 40ms. Further assume that there are on the average 2 additional expiring system timers. For a scheduling interval of 1 ms, the proportional maximum total scheduling overheads in percentage are listed in Table 7-4. Again the listed overheads are worst-case overheads which should not usually occur in a normal running operation. Even as such, these overheads should be acceptable.

***Table 7-4*** **Proportional maximum total scheduling overhead in percentage**

| System clock frequency | Number of real-time processes | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 5 | 10 | 20 |
| 120 MHz | 0.332 | 0.361 | 0.461 | 0.628 | 0.961 |
| 300 MHz | 0.199 | 0.217 | 0.277 | 0.377 | 0.577 |

In the case of timing violation and the need of installing a timing-violation handler, additional cycles are needed to install the handler. On PPDS, 49 instruction cycles are needed to install a timing-violation handler, which corresponds to 3.0625 µs for a 16 MHz system clock.

### 7.4.4 Other measurements

As is noted, most system management operations are executed either in the scope of a virtual real-time process or in the scope of a process of normal priority class 1.

Examples can be found in creating, starting and clearing processes and other house-keeping operations. These operations, as well as other functions executed in the scope of a real-time or non-real-time process, will not have a direct impact on the real-timeliness of the system.

As a reference, some measurements of them in the experimental PPDS environment are listed in the following table.

***Table 7-5*** **Execution overheads for some functions of the runtime system**

| Execution time in µs | -D USE_MACRO | | Compiler optimization -O2 |
|---|---|---|---|
| Function name | not set | set | |
| CreateProc_NORMAL | 373.2 | 339.6 | 224.9 |
| CreateProc_PERIODIC | 386.4 | 352.7 | 235.2 |
| CreateProc_VIRTUAL | 384.3 | 350.6 | 233.3 |
| StartProc_NORMAL | 22.6 | 19.3 | 18.9 |
| StartProc_RT | 66.6 | 48.1 | 45.5 |
| FORBID / PERMIT | 2.2 / 2.9 | 0.5 / 1.2 | 0.5 / 1.2 |
| DISABLE / ENABLE | 2.4 / 2.6 | 0.4 / 0.9 | 0.4 / 0.9 |
| Signal / Wait | 20.8 / 23.7 | 17.5 / 20.3 | 17.2 / 19.9 |
| ObtainSemaphore | 19.1 | 10.9 | 10.4 |
| ReleaseSemaphore | 17.4 | 10.7 | 10.6 |
| AllocMem | 64.2 | 49.0 | 45.1 |
| FreeMem | 56.9 | 42.0 | 39.6 |

## 7.5 Summary of the chapter

Orthogonal to simulation evaluations, experimental implementations and measurements have been conducted on a PPDS development system to further validate the feasibility of our soft real-time approach and algorithms proposed in the previous chapters.

Partly due to the restrictions of the hardware environment, the goal of the experimental implementation was not to implement a complete OS. Rather, the intention was to implement a run-time supporting system for a communication subsystem. However, the implemented run-time supporting system has provided many of the main functions of a normal operating system — in addition to a kernel, there are extended process management, memory management, dynamic lists management, semaphore management, signal facility and message facility. By implementing such a relatively complete run-tim supporting system, it is possible for us to gain useful insights into the realization of a soft real-time supporting system.

Basically, the CO-SCHEDULE and SIM-SCHEDULE soft real-time scheduling schemes have been implemented in the experimental system. For timing-overflow handling in SIM-SCHEDULE, both RELAY and RELAY-TRUNC methods have been implemented and can be used for different needs. The users can optionally provide a timing-violation handler which will be triggered to execute automatically when a timing overflow occurs. Programming interfaces for periodic real-time processes and virtual periodic real-time processes have been provided in such a way that the user of the processes only need to provide task body and specify real-time constraints such as start-time, period, execution-time. The real arrangement of the periodic instantiations of the task body is then done by the system automatically.

In our scheduler, the mapping of real-time and virtual real-time periodic processes to their priorities are dynamic and are done implicitly by the scheduling subsystem according to the GRMT scheduling theory. The scheduling of the high-priority periodic set is in the end conducted on the base of priority setting. The normal priority processes are scheduled according to the weighted round-robin method. In contrast to hard-real-time systems, admission control in a soft real-time system can be simplified to a large extent. For the experimental system, one of the two simple algorithms have been used to make admission control for processes requiring sure-guarantee and maybe-guarantee: either a mixed placement algorithm or a separate placement algorithm. To alleviate the problem of estimating the processing usage of a process, our idea is to make the scheduler function both as a timing enforcer and a timing predictor.

Measurements have been made to asses the performances of our experimental implementations. Results concerning basic scheduling overhead, additional overhead with full real-time support and other functions have confirmed that it is easy to implement efficient base schedulers incorporating our soft real-time scheduling and handling methods.

In short, the implementations and measurements have further verified the feasibility of our approach and algorithms. The practical experiences will also be very enlightening for more practical work in this direction.

# PART IV

# Complementary Techniques

# Chapter 8

# Predictable protocol processing

End-to-end QoS supports are essential for continuous media communications. As is partly described, adequate supports must be provided in the network as well as in the end hosts. Generally, protocols and mechanisms are needed to reserve required resources (buffers, processing capacity, link bandwidth, etc.) commensurate with the desired QoS of the applications, and resource scheduling and enforcing mechanisms are needed to satisfy the contracted QoS guarantees.

Without the appropriate support from the networks, it is very difficult to support CM communication on end-systems. Methods for supporting CM data transmissions and other real-time communications in networking systems are extensively discussed in the literature [Clark92, Ferrari90, Ferrari92, Delgrossi93b, Zhang93, Shenker95a, Shenker95b]. Above all, continuous media transmission requires the appropriate mechanisms from communication protocols [Dupuy92, Hehmann90, Moran92, Shepherd92]. Rate control, for example, is the most-often cited mechanism for CM transmission. Scheduling mechanisms within the network (on routers, etc.) are also needed to insulate the negative influences between communication streams and to achieve a high utilization of the shared communication media at the same time. (We would like to emphasize explicitly here that our schemes in this chapter *do not* assume that the networks can provide any rigid and absolute real-time support, although a better support from the underlying network will surely make the whole system more usable for a wider range of applications.)

Given the adequate support within the network to provide QoS guarantees, the communication subsystem within the end hosts must be made cognizant of the QoS requirements as well. This involves the support within the operating system and the networking software to conduct adequate link access, protocol processing, process scheduling, buffer management, communication service provision, etc. Given that the application-specific bandwidth guarantees can be established with the help of a reservation protocol like RSVP and that the modern network adapters (such as a Fore ATM adapter) have usually the capability to do traffic shaping so that stream sources at the network boundary are guaranteed to be QoS-conform to a large extent, link access has again been heavily related to the scheduling of communication processes as well as protocol processing.

The preceding chapters have presented a soft real-time framework with which all system activities on the end-system should be controlled to make the system predictable. It is the intention of this chapter to investigate how communication-related activities can be contained within the soft real-time framework so that the predictability of the system as a whole can be maintained. Since the processes accessing communication services are scheduled normally within the soft real-time framework, the main considerations will be made with regard to the protocol processing-related activities. Especially, the processing of network layer and transport layer protocols is traditionally regarded as part of the system kernel/server services. The main attention of the following discussion will be paid to the protocol processing for the network and transport layers.

In the following, the traditional protocol processing architecture and its shortcomings are analyzed. Methods are then given which will try to damp the effects of interrupts caused by communication activities. Application-driven protocol processing is then advocated as the feasible method to achieve predictable protocol processing. Other related issues such as the methods of communication service provision are also discussed. Again, the micro-monolithic kernel structure is usually taken as the reference.

## 8.1 Traditional protocol processing architecture and its shortcomings

The following takes the protocol processing procedure in the 4.3BSD UNIX system as a typical example of how the traditional monolithic OS conducts its protocol processing. The pros and cons of such a procedure are analyzed.

For transfer layer data communications in the traditional 4.3BSD UNIX system [Leffler89], the sending side behavior is generally under the control of a sending process where library functions and system calls should be invoked by the sending process and can be accounted to it. The problem lies mainly at the receiving side where the incoming packets are generally processed by the kernel at the cost of the whole system and then delivered to the processes destined.

Specifically, the protocol processing at the receiving side uses a kind of software interrupt technique (SOFTINT), in which protocol processing is done at a non-pre-emptable software interrupt level (a priority level higher than any user process but lower than the hardware interrupt level). As is argued in [Mercer91, Anderson91], SOFTINT technique has several advantages and also several disadvantages. On the one hand, SOFTINT technique avoids process-scheduling overhead while minimizing interrupt-masked time, it serializes protocol processing thus simplifying the synchronization mechanisms needed in the protocol processing modules. On the other hand, the SOFTINT technique is not appropriate for CM real-time communication. Software interrupts preempt time-critical processes, and may cause them to be blocked for unknown durations. Also, in the SOFTINT mechanism, packets are processed on a FIFO basis, which is undesirable for real-time preference and performance.

With the SOFTINT mechanism, protocol processing is begun right after hardware interrupt is ended. This inevitably incurs much disturbance on the ongoing user-level process which is interrupted by the interrupts resulted from the incoming data packets. In some system environments, the interrupt activities are so rare that the effects of the brief interrupt processing on real-time processes can be viewed as the stochastic variations of the RT processes themselves. In some other system environments, the interrupts constitute a main part of the whole system activities. The interrupt processing in such systems should then be done in a controlled manner, otherwise, the real-time guarantee and predictability of the system can not be managed.

Usually, several independent protocol stacks are supported in parallel by letting the data link layer network drivers dispatching the packets for different protocol stacks. The packets for different protocol stacks are interleaved somewhat but are generally still processed in a FCFS style. Packets for the same protocol stack are usually handled in the FCFS order.

The current implementations of some real-time protocol stacks used the conventional architecture detailed above. For example, the Tenet Protocol Stack (Suite 1) was implemented in the following manner [Barnerjea94, Ferrari92]:

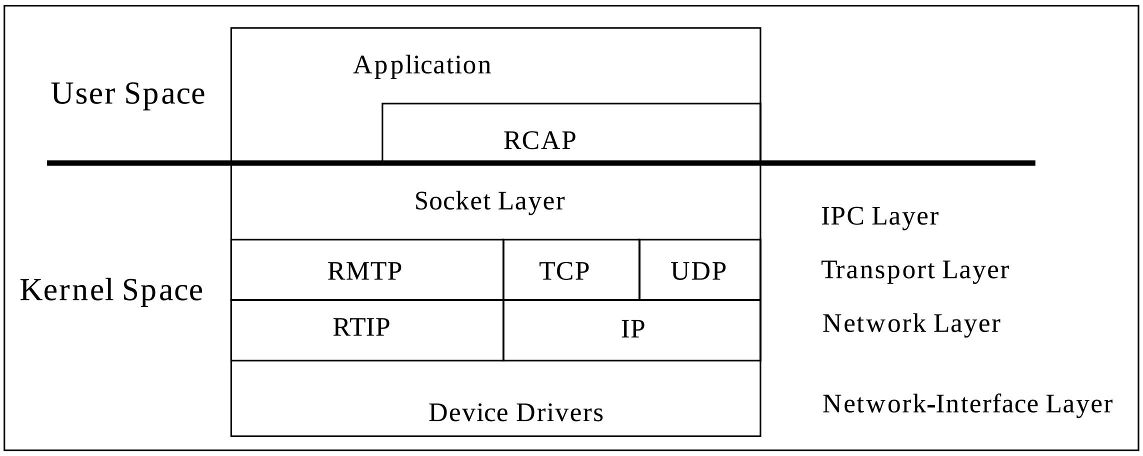

***Figure 8-1*** **Software Structure of Tenet suite and Internet protocols**

The same method had been adopted by the GMD implementation of the protocol XTP and the protocol stack XTP-lite / ST2 on a Sun platform [Fan93, Fan95f]:

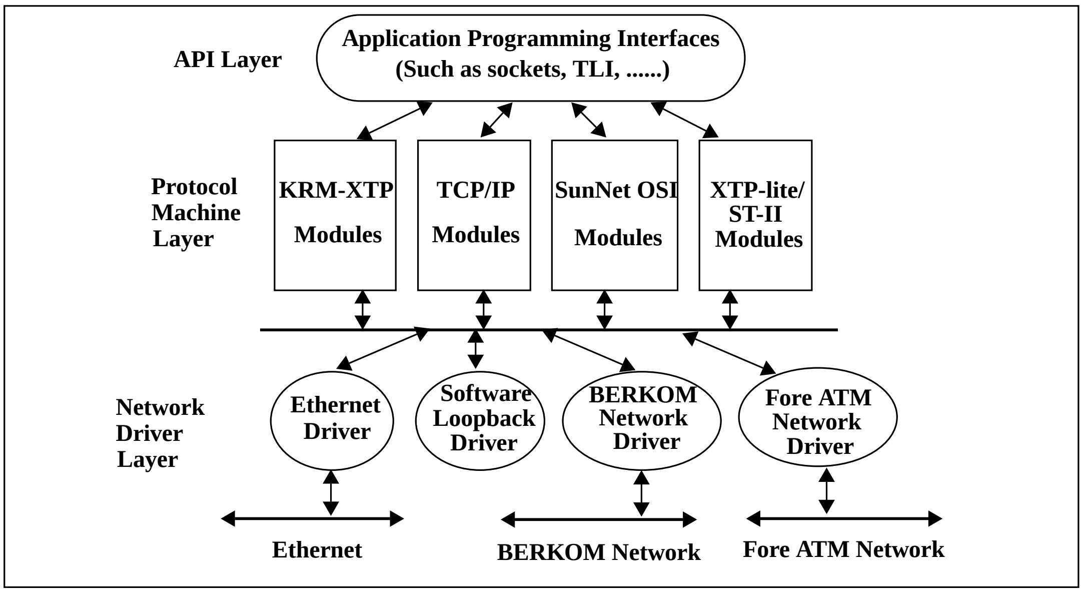

***Figure 8-2*** **Communication software structure on GMD-BTS**

These implementations used the conventional architecture and, therefore, had the same shortcomings detailed above: the interrupt processing and the following protocol processing caused by incoming packets preempt the current running user process and they are completed before any user processes have the chance to take over CPU again. The related processing costs are not reserved for or accounted to the processes which incur such processing. Thus, protocol processing is neither reservable nor predictable.

## 8.2 Damping the effects of interrupts

In any case, some processing capacity is needed for common path interrupt processing caused by incoming communication packets. In the case of non-trivial communication activities, such system overhead is too significant to be treated as system noises that can be tolerated in the soft real-time framework. As pointed out in Chapter 5, the overhead should be reserved in the form of one or more virtual periodic system-overhead processes.

This section proposes some measures which can be taken to reduce the influences and randomness of the interrupt processing caused by the incoming packets. The goal is to make the costs for such interrupt processing minimal and regular. It is then easier to closely reserve the costs in the form of some virtual periodic system-overhead processes.

### 8.2.1 Less interrupts and interrupts with regularity

No matter how the communication software is realized, it can not manage to keep a predictable and high performance when facing harsh system conditions such as inadequate I/O bus bandwidth, high I/O interrupt frequency and latency, unintelligent I/O processors, etc. Fortunately, it is usually possible to eliminate or alleviate these problems with the help of modern network adapters.

Though quite obvious, but many systems do not take into account such a simple fact: as long as the handling of interrupts is concerned, it is usually more efficient to not do it (too frequently and non-regularly) than to do it faster. In the case of continuous media communication where data volume is usually quite huge, such a consideration is of special importance.

Current RISC CPU's tend to have a high interrupt overhead resulting from saving a large register file and doing other stuff in software while RISC CPUs do them in hardware. It is thus even more important to reduce the number of interrupts and to make interrupt processing more regular. For CM communications, it is also possible to do so.

One common feature of multimedia communication is its periodicity. That is, the multimedia shows its periodic continuity both in the sense of time and space. The periodicity of the multimedia communication can be exploited to a large extent.

At the application user level, periodic sending, receiving and processing of data will result in a periodic scheduling of the corresponding tasks and threads. It is therefore possible to use the scheduling strategies which deal with periodic tasks to provide a certain degree of guarantee for their processing, as is exemplified in Chapter 5. The randomness of the multimedia communication is thus reduced to a large extent by the periodic scheduling of the communication processes and the rate-control exerted by the communication applications at the sending side (and possibly along the route).

At the lower level, interrupts resulted from multimedia communication can be partly spared if this periodicity can be used. By exploiting the periodicity at interrupt level, the processing overhead can be reduced and principally controlled. This is of special importance since the low-level interrupts have usually taken a large part of the processing capacity of an ordinary workstation.

Several methods might be used to reduce interrupt processing overhead. In one case, for example, an interrupt can be generated with a periodicity of a certain $n$ microseconds so that the incoming packets accumulated during this time period can be processed together. In another case, a cumulative interrupt can be used so that an interrupt will be generated for every $m$ packets at a network interface instead of for every packet. Or, the network interface might be enabled to generate an interrupt if there are some packets at the interface or when a timer runs out. In this way, it can be prevented that the interface generates interrupt too late because there are not enough packets arriving in the last timer period. In yet another case, the network interface does not generate interrupts at all. It will be the network protocol module itself that polls the interface periodically.

Consider a concrete example [Fan95c] in our ATM LAN environment which is based on the hardware from Fore Systems Inc. For data transmission between two Sun workstations using UDP/IP over the ATM LAN, the receiving side is generally too slow where the connections virtually collapsed due to the non-existing flow control and due to the inability of the CPU to both reassemble the incoming ATM cells and to forward the messages upwards the protocol stack in time. In particular, the data delivery is not able to provide a higher bandwidth than 45 Mbit/s to the network layer (symptom: “packet dropped, IP queue full”) and 18 Mbit/s to the application (symptom: “socket overflow”). One main reason is that the numerous interrupts caused by incoming ATM cells took too much CPU time.

A reduction of interrupt processing is clearly necessary for such environments. The above-mentioned several methods can be used in some cases to reduce interrupts and to make interrupts regular. They have actually also been made possible by current network interface technology. Presently, some network interfaces have already provided some hardware support for realizing the afore-mentioned methods. In the newer version of the Fore ATM network interface, for example, one can have the ready support for cumulative interrupt generation, time-out interrupt generation, etc. [FORE93, FORE95]. The experimental network interfaces like “Afterburner” had also similar features.

### 8.2.2 Physical reservation and separation

The burden induced by heavy communication can also be partly eliminated by appropriate extra hardware support to reach the effect of physical reservation and separation.

DMA is usually used when the I/O board’s memory is on a separate bus and is isolated from the host’s memory. Here, a DMA controller (DMAC) is used to transfer data between the host memory and the I/O board memory (see the following figure). The protocol processing board can be used in a similar configuration.

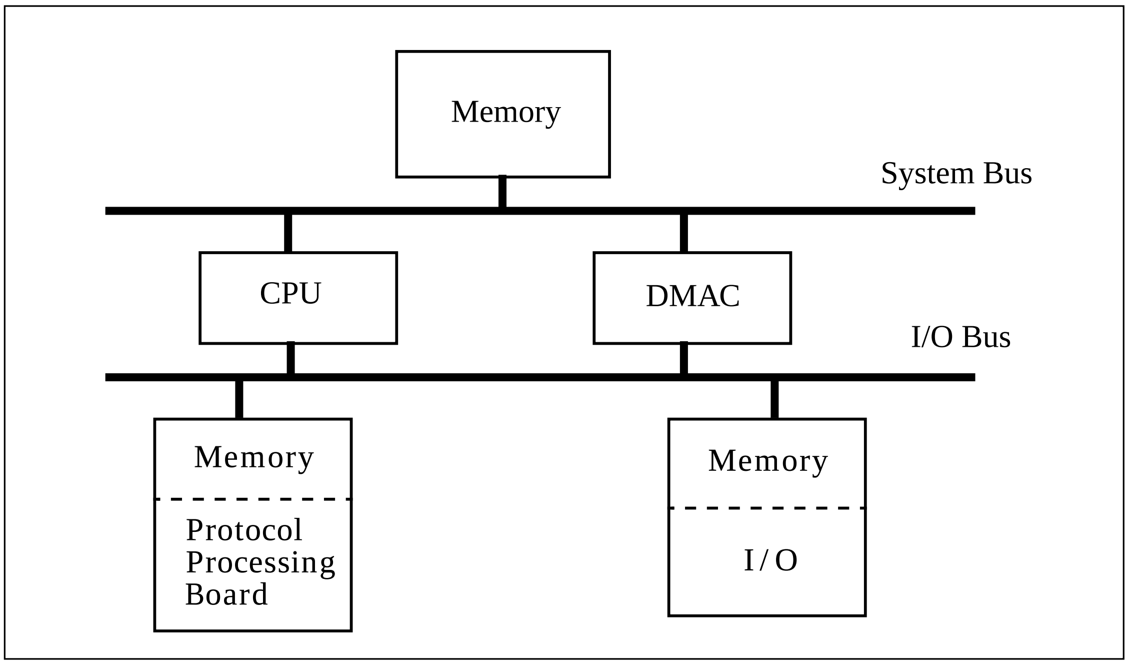

***Figure 8-3*** **Front-end protocol processing board**

As a typical form of physical reservation, the front-end subsystem structure has its advantages as well as its disadvantages. Actually, the feasibility of a front-end subsystem depends a lot on the intended application scenarios.

On a front-end subsystem, a relatively independent MM supporting system can be built. Different and hopefully more suitable OS or run-time environments can be used on the front-end and on the host. The performance of the whole system can be possibly enhanced through the extra processing power of the front-end and through the parallelization of processing of the front-end and the host. But the question is, to what extent are the anticipated advantages realizable if the interaction between host and front-end is unavoidable.

Through the development of the MNI front-end, for example, the Distributed Multimedia Research Group at Lancaster University had come to the conclusion that the use of a front-end structure is a partial success [Blair92a]. In the Bermate project at GMD-FOKUS [FOKUS92], a front-end subsystem had also been experimentally used as an extra place for protocol processing and multimedia application. The goal of performance improvement and parallelization was also achieved to some extent. A reintegration of the front-end subsystem into the host itself will possibly meet the close interaction requirements between host and front-end for some applications.

### 8.2.3 Two-level interrupt processing

As pointed out in Section 8.1, the SOFTINT mechanism used for interrupt processing in the traditional protocol processing architecture has the following shortcomings: the interrupt processing and the following protocol processing caused by incoming packets preempt the current running process and they are completed before any user processes have the chance to take over CPU again. The related processing overheads are not reserved for or accounted to the processes which incur such processing. For the systems where communication activities are no rare events, such protocol processing is neither reservable nor predictable.

For the communication-intensive systems, it is especially important to be able to reserve the processing capacity for communications and to be able to make accounting on the capacity used by different processes for their communications.

Note also that in these systems, the incoming packets from the networks generate a large quantity of interrupts, which should be handled in time otherwise the packets will be overridden by newer incoming packets at the same interfaces and be lost. Therefore, on the one hand, the interrupts should be responded quickly in order to avoid packet loss. On the other hand, the interrupt processing related to packet incoming should be done in a controlled manner in order not to disturb any current real-time activities too much.

Principally, the interrupts caused by communications or other activities shouldn't be allowed to steal away at will the processor from real-time applications which have previously reserved a share of the processor. Here, we can borrow an idea used in developing device drivers [Armand91]. A two-level interrupt scheme can be used to achieve this effect. At the lower level, the interrupt handlers do no other processing other than just record the occurrence of an interrupt then disable the cause. The scheduler will then make the right decision about next runner after that.

Thus, the main processing of the interrupt itself is deferred to a later, more appropriate time. Note again that a characteristics of the continuous media communications is exploited here: the CM data packets need not be processed as soon as possible, they need only to be processed in time — before their playback points.

For interrupts caused by incoming packets, the first-level interrupt processing will simply queue the incoming packets and clear the interrupt cause. Since in modern MM systems, the relatively independent network interfacing subsystem will usually allocate the memory for storing the incoming packets and will give only the reference to the packets for further processing, this first-level processing will therefore be a very simple operation: enqueue the references of the new packets to waiting queues which will be processed by protocol processing later. In the next section, we will further discuss when and how the actual main protocol processing triggered by these incoming packets should be carried out.

Using this scheme, the disturbances caused by non-deterministic interrupt events can be kept as low as possible.

## 8.3 Structuring protocol processing

According to the above two-level interrupt processing scheme, the main protocol processing should be conducted in the second level and should be scheduled in the framework of real-time processing. Two methods can possibly be used to structure this part of protocol processing: multithreaded protocol processing or application-driven protocol processing.

### 8.3.1 Multi-threaded protocol processing

In the traditional protocol processing scheme using a mechanism like SOFTINT, the whole protocol processing is done at a non-preemptable software interrupt level. Protocol processing is begun right after hardware interrupt is ended. This has the same effect of serializing all incoming packets in a single processing thread. A multithreaded protocol processing structure can be used for the second level of packet interrupt processing.

Actually, multithreading protocol processing has already been advocated in [Anderson91, Mercer91, Hutchinson91, Wolf93] in which protocol-processing activities are implemented as independent, prioritized threads with / without preemption possibility. In our opinion, associating messages to as many threads as possible, as in the scheme advocated in [Hutchinson91, Wolf93], is not advisable in MM case. Such fine-grained associations are difficult to manage and schedule. For MM communication, connection is a natural unit for ordering messages (we argue that connectionless service is not feasible for MM communications where QoS related to a connection should be somehow supported). It seems reasonable to associate the messages belonging to a connection with the processing thread for this connection. Resource reservation, including processing capacity reservation, can then be done in a connection-oriented manner. Note that connection establishment and release are relatively complex and time-consuming. They could be treated as non-real-time and system specific, and they could be conducted in a different thread than that for the normal transmission path.

The multithreaded processing model can be implemented in the form of a multithreaded server on top of a micro-kernel structure or in a multithreaded monolithic kernel. For the case of a multithreaded kernel, the main advantage is that the entire protocol subsystem just happens to run in a single privileged mode and can be more efficient. The main disadvantage is that there is no good protection between protocols or between protocols and other system modules, since they all run in the same kernel address space.

As far as the reservation of processing capacity is concerned, the multithreaded protocol processing has its problems. It is difficult to define a feasible interface which can be used by an application to ask for the reservation of a protocol processing thread on its behalf. It is also difficult to arrange a joint admission control of the application and the reserved protocol processing thread. In view of these difficulties, the following advocates a more direct application-driven protocol processing method which can be used with our micro-monolithic approach easily.

### 8.3.2 Application-driven protocol processing

We suggest to use an application-driven protocol processing model as an alternative to structure protocol processing in the second-level packet interrupt processing. The model can be briefly explained as follows: in the first level of packet interrupt processing, packets are queued to different connection-oriented packet queues. The processing of the packets in the queues will then begin before the application possessing the connection will possibly begin to use the packets. From a logical point of view, the (network layer and transport layer) protocol processing related to the data packets is just part of other processing to be applied on these packets such as decoding etc. At the sending side, the protocol processing can also be viewed as a logical part of processing and forming a data packet.

For this method to work, we should, of course, be able to find connection identification (session identification) at low level. For protocols like TCP/IP and XTP, the connection identification is easy to find by “looking into” the packets — TCP/IP packets have “port number”, XTP packets have “key”, IPv6 packets have “flow label”. For other protocols like Tempo++, a connection-oriented network service with some identification is also assumed. Over an ATM network, the connection identification is simplified in the form of a virtual circuit identifier, i.e., the VPI/VCI combination.

In this manner, processing capacity reservation can be made mainly in an application-oriented manner, not scattered over various parts of user and system activities. That is, the time used for main protocol processing is then part of the application processing time and the schedulability test will be done by treating protocol processing consumption as part of application processing consumption.

There are several effects of the above “lazy-evaluation” style of protocol processing at receiving side: (1) Interrupt processing is done in a controlled manner so that the real-time processes under current execution will not be disturbed too much; (2) Process reserve their own processing capacity, including the capacity for protocol processing, so that these processes can be scheduled in a real-time manner.

The most direct method to implement the above application-driven protocol processing scheme is for the user processes to do the main part of protocol processing through the help of protocol processing libraries. (It might also be possible for a user process to relay the processing to a system process which runs using the timing budget of the user process. But this has the same complications as the above multithreaded protocol processing method.) Such a method also makes it very simple and direct to reserve the capacity needed by protocol processing as part of the whole user processing capacity. Recall again that techniques and measures have been taken to reduce the influences and randomness of the first-level interrupt processing caused by incoming packets so that the cost for such interrupt processing can be more closely reserved in the form of the virtual “system-overhead” periodic processes mentioned before.

The related idea of application-level protocol processing was originally proposed under the name of “application level framing” as an approach for improving performance of protocol processing [Clark90]. The idea was later verified in many projects as a feasible method to do protocol processing in the user space and to improve flexibility as well as performance at the same time [Thekkath93, Maeda93, Edwards94]. Here, we argue that it is also applicable for the easy reservation of protocol processing capacity and for the improvement of the predictability of protocol processing.

## 8.4 Other related issues

There are a few other issues which should be briefly dealt with.

### 8.4.1 Methods of communication service provision

Generally, the interaction relationship between the communication service user and the service provider can be differentiated between explicit interaction and implicit interaction.

- Explicit interaction modes

The explicit interaction modes can be one of the following three: (a) Service-user-active mode: service users issue system call and then retry or be blocked; (b) Service-provider-active mode: service providers upcall the consuming function of service users in receiving; (c) Semi-provider-active mode: service providers notify the consumers through a way like signal but the providers do not have control over when the signal handlers can be activated.

For irregular interaction and monolithic kernel, mode (a) is the easiest way to implement and is also quite efficient. Upcall is the fastest reaction method for receiving. Mode (c) might be naturally done as an extension of the existing OS systems.

The information exchanged between communication service user and provider can be provided by shared buffer, through memory remapping or through explicit parameter delivery. These interaction modes are sometimes desirable because they are easy to use.

- Implicit timing relationship

For the non-explicit interaction modes, the timing relationship is implicitly arranged. Shared control information between communication service user and provider is necessary for all cases. Data can also be exchanged by way of shared buffer or memory remapping. Synchronization variables and mechanisms should be used to maintain integrity of data.

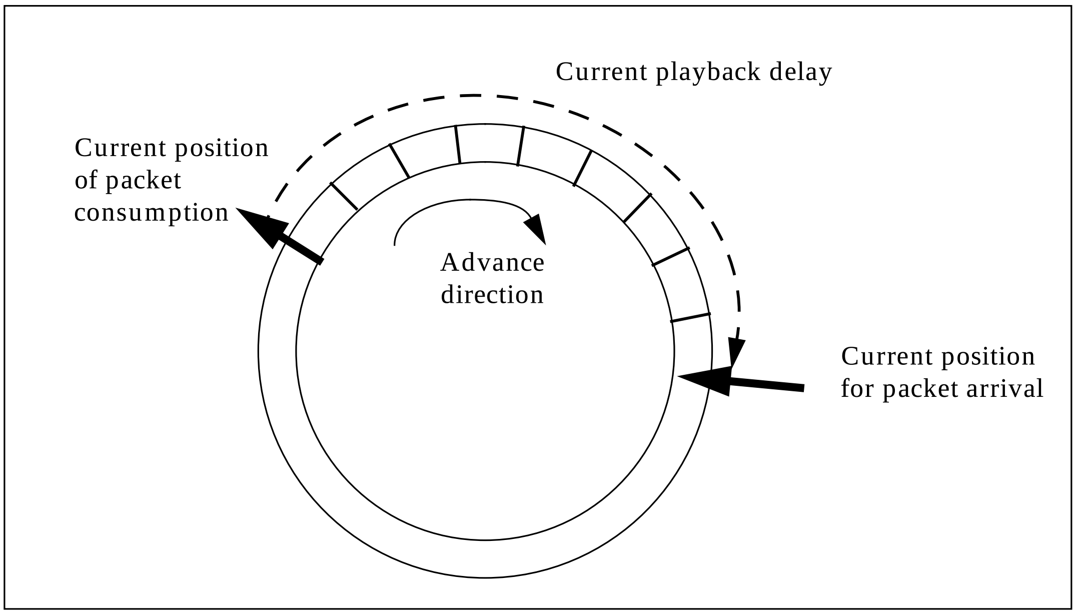
Current playback delay
Current position of packet consumption
Advance direction
Current position for packet arrival

***Figure 8-4*** **Use of ring-buffer for implicit timing relationship**

In our opinion, the implicit timing relationship is suitable and efficient for an interaction in good regularity. This is the reason why some MM systems advocate the use of such a scheme in the form of a ring buffer [Wolfinger92, Kihara93], where the timing relationship between the producer and the consumer is implicitly regulated by the current positions for new arrival and consumption.

### 8.4.2 Granularity of timing

It is clear that the basic timing facilities in a MM-supporting OS should have higher precisions than in a non-RT OS in order to make the timing results meaningful.

As for protocol processing, the scheme used in [Anderson90a, Hagsand94] associates each message with its own deadline. Such a scheme is direct and easy to apply but it incurs a high bandwidth and processing overhead.

In our opinion, there are also other applicable views of timing granularities. The CM applications usually posses periodicity, thus it is possible to define a single logical group deadline for all the messages in the period of some logical units. The timing granularity is then larger then the message-oriented timing granularity and it may reduce the overhead in scheduling and bandwidth. An effect of larger timing granularity is the possible larger “burst” of processing or delay jitter.

### 8.4.3 Complications with a strict micro-kernel model

The discussions in this chapter have taken mainly the micro-monolithic kernel structure as the reference OS structure, since it is the advocated OS structure best suitable for the realization of the soft real-time framework. Other than this, a few words on the complications with the protocol processing by a strict micro-kernel approach might be necessary.

As many other system services, the communication services are generally provided as separate communication server in a micro-kernel OS. As is well known, different system servers are of different protection domains and are embodied in different system processes which communicate which each other and application processes via the micro-kernel using message passing. The related advantages of robustness, extensibility and configurability come at some cost in performance since invoking a service now requires communication between two processes. In the micro-kernel system, this overhead includes two context switches as opposed to a simple processor trap in a (micro-)monolithic kernel system. In addition to the performance consideration, the reservation and accounting related to the communications in the micro-kernel system are also more complicated since these also involve multiple protection domains.

Actually, the method of moving the protocol processing into application processes is now also advocated in the micro-kernel system to solve all the above complications [Maeda93, Mercer94b].

## 8.5 Summary of the chapter

Given the adequate support within the network to provide QoS guarantees, the communication subsystem within the end hosts must be made cognizant of the QoS requirements as well. The intention of this chapter is to investigate how communication-related activities can be contained within the soft real-time framework so that the predictability of the system as a whole can be maintained. The main attention of the discussion is paid to the protocol processing for the network and transport layers, which is traditionally regarded as part of system kernel/server services.

Methods have been investigated which can be used to reduce the influences and randomness of the interrupt processing caused by incoming packets. They either exploit the characteristics of continuous media communication to make interrupts less and regular, or, simply make use of the possibility of physical reservation and separation. Two-level interrupt processing is also advocated where the first-level packet interrupt processing is made brief. The use of these methods can make the costs for common-path packet interrupt processing minimal and regular. For systems with non-trivial communications, some processing capacity is still needed for common path interrupt processing caused by incoming communication packets. It can be done in the form of reserving some virtual periodic system-overhead processes.

In the two-level interrupt processing scheme, the main protocol processing is conducted in the second level and is scheduled in the framework of real-time processing. Although it is possible to structure the second higher level packet interrupt processing in a multithreaded manner, we advocate a more direct application-driven protocol processing scheme which can be used with our micro-monolithic approach easily. The most direct method to implement the application-driven protocol processing scheme is for the user processes to do the main part of protocol processing through the help of protocol processing libraries. Such a method also makes it very simple and direct to reserve the capacity needed by protocol processing as part of the whole user processing capacity and to do the related accounting.

A few other related issues such as methods of communication service provision and granularity of timing are also briefly dealt with.

In short, the methods in this chapter can be used to bring the protocol processing activity, which is one of the main activities on a multimedia communication system other than the normal application processing, under the predictable control of the soft real-time framework.

# Chapter 9

# Adaptive service provision

As argued above, a multimedia application system need to be predictable as well as flexible. Soft real-time techniques can not only provide some degree of soft guarantee but can also accommodate the flexibility of adapting to changing system environments. The latter takes the form that it is possible to support a flexible and adaptive service model in the soft real-time framework.

In our system, the support of Quality of Service (QoS) requirements of applications are realized through micro-level mechanisms and macro-level strategies. As described in the previous chapters, a set of micro-level OS mechanisms are needed to support the implementation of a processing and transport system that is predictable enough to make various degrees of guarantees possible. At a higher macro-level, we propose to use a **f**lexible and **a**daptive **s**ervice suppor**t**ing model (FAST model) to provide OS services to support the flexible and adaptive nature of many multimedia communication applications.

Here, we would like to emphasize explicitly once more that we *do not* promote a system that relies exclusively on “graceful degradation” to attempt to handle system overload, as is done in [Compton94, Fall95]. Instead, the algorithms and techniques proposed in the preceding chapters have laid a foundation for building soft real-time supporting systems that are quite predictable under general and normal operations. Our adaptive service model aims primarily at serving macro-level user-initiated adaptation and supporting-system-initiated adaptation, both of which are still directly supported and predictively controlled by the supporting system, as will be described in detail below.

This chapter is divided into three sections. They deal with the applicability of the model, the description of the model and the realization aspects of the model respectively.

## 9.1 Applicability of the model

The section deals with the applicability of the proposed service model. First of all, the necessity for supporting a flexible and adaptive service model (the FAST model) by our OS supporting framework is given. We also analyze the adaptive nature of many multimedia applications that can be easily supported by the FAST model. The technical bases for realizing the adaptive MM communication applications and our proposed soft real-time OS support for the FAST model are then sketched briefly.

### 9.1.1 Why an adaptive service model

The need for adaptability comes roughly from two categories. Either, a continuous media-related application will actively switch its working modes and initiates the changes in its QoS requirements itself. Or, a multimedia communication application has to passively tolerate the changes in their environmental parameters to some extent. The adaptation requirements can be correspondingly classified as application-initiated and supporting-system-initiated. The CM supporting system, especially the transport subsystem, should be able to cope with both situations.

First, let us have a look at the cases where the applications have to change / adapt its “working modes”.

Since the resources that a user can use is always limited in one way or another, it is sometimes the responsibility of the user to decide how he will divide the usage of the resources for different subtasks of his whole application. Again, consider a multimedia collaboration (MMC) application scenario. For some time period, the users might want to lower the interactive audio / video quality in order to achieve a faster exchange of working documents. After that period, the users might again want to raise the quality of A / V exchange and make document exchange less significant. (The switch between these “working modes” might occur semi-automatically in a concrete MMC software, see *Section 9.3.6 Automatic higher-level adaptation management*.) Note that the whole resource capacity (CPU power, bandwidth) is generally constant, the change of "working modes" will inevitably make it necessary to adapt the "working modes" of MMC components. It follows that these MMC components should be made adaptive and that the supporting system should be able to adapt the resource partition / allocation and management correspondingly.

By the way, the adaptability is also useful even when an application has no intention to change its working mode at run-time. The reason is that it is usually difficult to do an exact mapping from user-level QoS parameters to lower-level system QoS parameters such as exact bandwidth and CPU usage capacity. According to the actual parameters reported by the system, it would be very helpful to do an adjustment of these mappings in the initial phase of the application.

When, then, does a supporting system have to initiate the adaptation of the CM communication applications?

This can happen, for example, when the activities supported by the supporting system have changed significantly. For example, the supporting system has to accept and process more connections or the network service has been degraded because of congestion or fallout of some routers. In such cases, a repartition / reallocation of the available resources is necessary and the applications should adapt correspondingly.

Two aspects make it possible to use an adaptive service model: the adaptive nature of some CM applications and the technical bases that supports adaptability. The latter includes media coding and media scaling, communication protocol mechanisms and other OS support. We argue that flexible soft real-time OS supports should be provided to make an adaptive system support complete. And some proposals for providing such OS supports are made.

### 9.1.2 Adaptive nature of CM applications

The adaptive nature of many continuous media applications has been partly discussed in *Section 3.2 "Soft" features of CM applications and communications*. We have pointed out that the non-rigidity or flexibility of CM applications and communications can be seen in several aspects. One aspect is that we can identify three kinds of possible adaptabilities which can be exploited. That is, rate, data volume and playback delay of CM applications are in many cases adaptive and can be adjusted in a certain range.

In short, the adaptive nature of human users' perception to continuous media has lead to the adaptive nature of many MM applications. An adaptive MM application can function with different degrees of QoS requirements and corresponding supports.

### 9.1.3 Technical bases for implementing adaptive CM applications

In the current technical literature, there are already many examples of the implementation of adaptive CM applications [Clark92, Kurose84, Partridge91, Mathur93]. Several aspects of technical bases make the implementation of the adaptability of these applications possible.

For one thing, flexible coding methods are used to produce adaptable data volumes and data rates for different media streams. For another thing, communication protocol mechanisms are used to negotiate and renegotiate the QoS agreements among communications partners.

In addition, OS mechanism support is necessary. In our opinion, the OS supports for the implementation of these adaptive applications are currently still not enough or still need to be improved. OS support is needed so that the time-constraints of the application processes can be adapted. OS support is also needed in monitoring and adjusting all the other system activities, especially the activities in the transport subsystem.

Together with a treatment of the technical bases for implementing FAST model itself, the treatment of the above-mentioned technical bases for implementing adaptive applications will be given after a more concrete description of the FAST model. This will include especially our OS mechanism proposals and implementations.

## 9.2 FAST Model description

This section describes the components of the FAST model and argues for its advantages.

### 9.2.1 Components of a flexible and adaptive service supporting model

The main purpose of the **f**lexible and **a**daptive **s**ervice suppor**t**ing model (the FAST model) is to support both the application-initiated adaptations and the supporting-system-initiated adaptations.

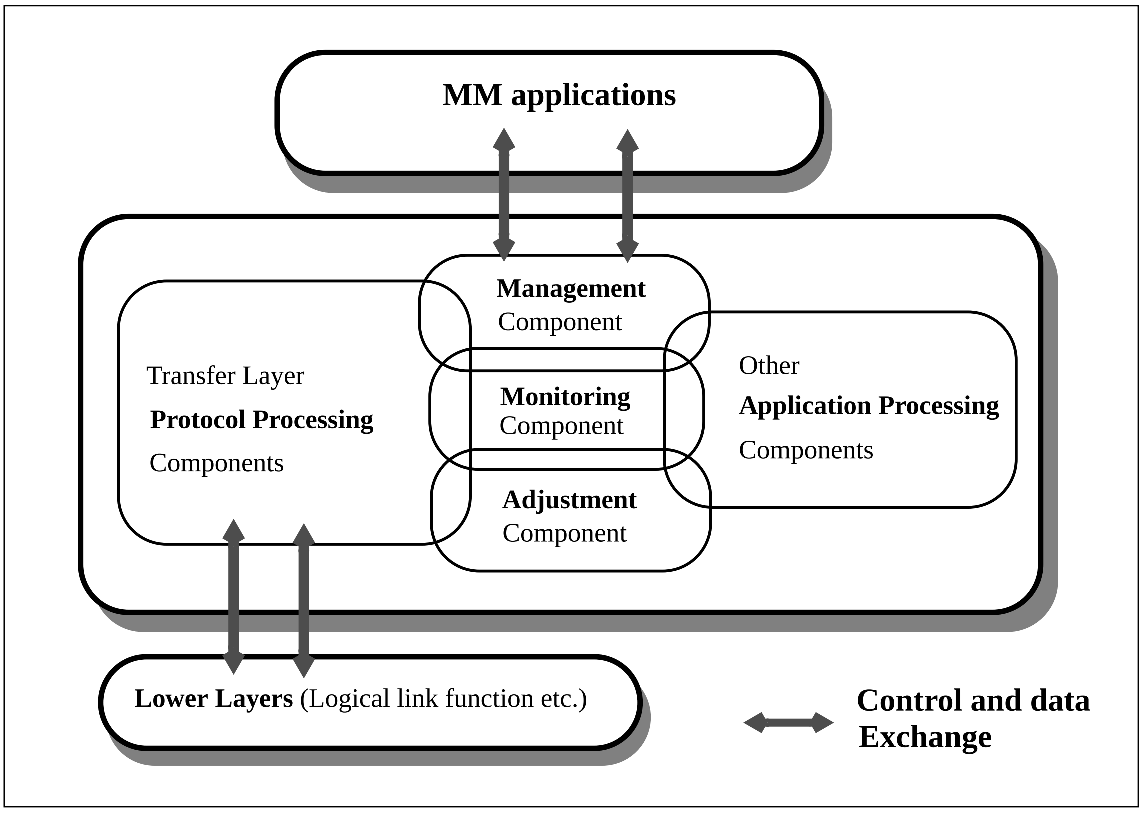
MM applications
Management Component
Transfer Layer Protocol Processing Components
Monitoring Component
Other Application Processing Components
Adjustment Component
Lower Layers (Logical link function etc.)
Control and data Exchange

*Figure 9-1* **System overview of the FAST model**

A system overview of the FAST model is shown in the above figure. The construction of the (supporting) system, especially the transfer layer (including network layer and transport layer protocol processing), will include not only normal processing components but also management component, monitoring component and adjustment component. Boundaries between these components shown in the figure are only schematic. They may not be so clear and absolute in a concrete implementation.

Let's look at an example of the supporting-system-initiated adaptation. Assume a live video transmission, the scenario will then work like this: the upper layer applications give their requirements in the form of a set of QoS parameters through the service supporting interface to the transfer layer. The management component will check these QoS parameters and try to allocate resources for the satisfaction of them. If the satisfaction lies in a certain confidence range (not necessarily a 100% absolute guarantee), the service requests can then be accepted and serviced. During the transmission period, however, the actual fulfillment of the QoS parameters will be monitored by the monitoring component. In case of a QoS violation or unsatisfaction, the upper layer will be notified of the current situation. The upper layer can then decide whether to abort (video quality should not be worse), to continue (video quality is still bearable) or negotiate new QoS parameters and then continue (maybe change to a lower resolution or a lower frame rate of video). The adjustment component and management component should do the necessary adjustments if necessary.

During different periods of the application execution, the upper layer application might actively initiate the changes of video transmission in terms of resolution or frame rate. These changes can also be supported by the management component and the adjustment component. And this constitutes an example of the application-initiated adaptation.

In the traditional service provision model, the support of QoS parameters by the scheduling subsystem and the transfer layer is somewhat passive. In the FAST model, the processing and trasnport subsystem and its user (higher layer protocols and applications) will interact with each other during the whole process. That is, both service provider and service user are active participants of the whole process. Both can react to changes actively. This property of activeness applies also to the cases where no network communications are involved.

### 9.2.2 Advantages of the FAST model

- **FAST model is adaptive, flexible and accommodative**

Based on the assumptions that a perfect scheduling and resource reservation is not always feasible, the monitoring component is constructed to compensate them. That is, monitoring methods can be used in the case of possible violations of QoS agreements. Note the function of monitoring and the reaction to related violation is two fold: it can monitor the violation from the side of service user so that the effects of the misbehaving users can be constrained; it can at the same time monitor the real achievements of the service providers so that either the corrective actions in the scope of service providers can be undertaken or some indications of such violations can be delivered to the service users so that the service users may undertake possible corrective actions.

As can be seen from the above scenario, one advantage of the FAST model is that it can satisfy the need for adaptive service support (adaptability). At the same time, it has also the advantage of being able to accommodate various degrees of service requirements and service provision strategies (flexibility). That is, statistical and optimistic-style service supporting strategies which provide various degrees of soft guarantee can be used since monitoring and adjustment component will come to help in case of violation of QoS fulfillment. Pessimistic-style, full-reservation-and-hard-guarantee service provision strategies can also be accommodated under this model since it simply means QoS parameters will always be observed and the monitoring and adjustment functionality can virtually be turned off.

- **FAST model is practicable**

The FAST model does not base on any assumptions of perfect scheduling or resource reservation. Rather, it assumes that a perfect and absolute guarantee is not always possible or feasible. The imperfection will be compensated by the adaptability of the whole supporting system and the higher layer applications.

Because the adaptive nature of the FAST model will be able to compensate the case where the expected degree of availability can not be maintained, the estimation of availability of the resources can be made in an optimistic manner even in the case where the “calculation” and “reservation” of some resources are vague or difficult.

For example, it is often very difficult to calculate precisely and reserve exactly the processing power under currently available operating systems. For service supporting schemes which are based on the assumption that an absolute guarantee of processing power should be available, this will cause a very big practical problem. They have to make very pessimistic estimations in these cases and this will inevitably lead to an over-reservation and resource waste. For FAST, instead, an estimation of processing bound based on certain degree of confidence can easily be found and used.

This means that some simple strategies, mechanisms and algorithms can and should be used in the realization of the FAST model so that the advantages of real-timeliness, predictability and controllability can still be achieved in a dynamic manner. For any concrete implementations, this starting point is of course very important.

## 9.3 Realization aspects

This section describes in detail the various aspects of the realization and application of the FAST model. It gives examples of media scaling and transport QoS negotiation mechanisms which can be used in implementing adaptive multimedia communication applications. It also describes the needed OS support for realizing the FAST model. As an emphasis, the detail of soft real-time scheduler support in adapting the variance of the time-constraints of the processes are provided.

### 9.3.1 Issues in realizing and applying the FAST model

The FAST model provides a framework for providing OS services to multimedia communications and applications. At the same time, the FAST model also provides a framework for the MM communications and applications to use OS services. Surrounding this service supporting model, the following issues need to be investigated in order to make it possible to realize and apply the FAST model:

(1) Service provision and access structure should be designed so that QoS negotia-

tions can be made and QoS violations can be indicated; (2) Resource management and reservation methods should be designed and implemented so that the QoS parameters can be really maintained; (3) Monitoring possibilities and methods should be investigated to prevent and detect violations on QoS agreements; (4) Adjustment possibilities and methods should be examined to see to what extent the adaptability in the scope of transfer layer and other system components is possible.

- **A service model best supported by a soft real-time OS**

As a flexible service-supporting model, the FAST model can principally be utilized in almost all OS environments to a more or less extent. But we argue that the model should at best be supported by a soft real-time OS in order to achieve the optimal effects, since the implementations of monitoring, adjustment and adaptation functions can be best realized in a soft real-time framework. As presented in the preceding chapters, a feasible predictable protocol processing architecture and various degrees of soft guarantee of multimedia activities can be relatively easily provided in a soft real-time environment. In addition, the real-time environment has usually a more precise timing support, which is the basis for all monitoring functions. With the support from soft real-time scheduler, the monitoring and adaptation of real-time processes can also be easily provided, as will be presented in detail in *Section 9.3.4 Adaptation support from OS scheduler*.

In the following, we will have a look at several areas which are vital for the realization and application of the FAST service supporting model. The areas include media coding and media scaling, support from communication protocol mechanisms, support from OS scheduler and other OS resource pools. In traversing each area, we will deal with the issues listed at the beginning of this subsection, as long as they are relevant. Concrete design and implementation examples are used to validate our claims.

### 9.3.2 Media encoding and media scaling

First of all, the presentation forms of multimedia should be adaptive, otherwise, the applications using them can not be made adaptive. This is indeed the case.

Current media encoding and compression methods for audiovisual data make it possible to “scale” the multimedia data streams to a more or less extent [Clark92, Steinmetz93, Delgrossi93a, Partridge94]. By media scaling, we mean the practice of subsampling a multimedia data stream and presenting only some fraction of its original contents. Generally, media scaling can be done at either the media source where a media stream stems or at the media sink where a media stream is consumed. At the source, for example, frame rate can be scaled up and down. While at the sink side, a hierarchical decoding method can be applied.

Scaling methods used in a multimedia transport system can be classified as transparent and non-transparent [Sandvoss94]:

Transparent scaling methods can be applied independently from the upper protocol and application layers. That is, the transport system has the possibility of scaling the media stream on its own. Transparent scaling is usually achieved by dropping some portions of the data stream. These portions, in the form of single frames or substreams, need to be identifiable by the transport system.

Non-transparent scaling methods require an interaction of the transport system with the upper layers. This kind of scaling implies a modification of the media stream before it is given to the transport layer. Non-transparent scaling methods typically require the modification of some parameters of the coding algorithms or even recording of a stream that was previously encoded in a different format.

- **Scaling examples of audio/video data streams**

For audio data, non-transparent scaling can be easily done by changing the sampling rate at the audio stream source. Transparent scaling is usually more difficult for an audio data stream, since presenting only a portion of the original audio data is easily noticed by a human listener. Possibilities still exist. For example, voice can adapt imperceptibly by adjusting silent periods. Dropping a channel of a stereo stream is another example.

For video streams, the applicability of a specific scaling method depends strongly on the underlying coding and compression technique. Generally, there are several possible domains of a video signal to which scaling can be applied. This leads to the possibility of temporal scaling, spatial scaling, frequency scaling, amplitudinal scaling and color space scaling.

An example of the temporal scaling of a video stream is the variation on the number of video frames for transmission within a time interval. This is best suited for video streams in which individual frames are self-contained and can be accessed independently. An example of the spatial scaling is the change of the number of pixels of each image in a video stream. One way of color space scaling is to switch between color and greyscale presentations.

An important form of scaling technique is layered encoding, where the stream is composed of various substreams with different importance / quality significance. For a spatially scaled stream, for example, a substream might consist of odd / odd pixels while the other substream would consist of even / even pixels. Exemplified by layered MPEG, a video stream could use one substream for intra-coded frames, which can be independently decoded and are themselves self-contained, and one or even more substreams for other frames.

Note that the varying amount of the audio visual data resulting from dynamic media scaling has a direct impact on the network resources, such as bandwidth, needed to transmit them and on the end system resources, such as processing power, needed to process them.

### 9.3.3 Support from communication protocol mechanisms

The implementation and the use of the transfer layer (including OSI network and transport layer) according to the FAST model is critical for supporting adaptive multimedia communications and applications. To provide adaptive communication services with soft guarantee, we need not only the protocol mechanisms to conduct QoS parameter negotiation and renegotiation, but also the protocol mechanisms to support the implementation and monitoring of the negotiated communication QoS parameters.

Many current protocols have integrated mechanisms to support communication QoS. These include the negotiation and renegotiation of QoS parameters and the maintenance of these parameters during the period of communication. We note, of course, that the realization and maintenance of these QoS parameters, especially the performance parameters, can not be done by the protocol processing alone. The support from the underlying networks, which should be able to provide and adapt the basic networking performance parameters to some extent, and the support from operating system functions are necessary or even decisive. In an ATM network, for example, the renegotiation of QoS parameters over virtual channels (VCs) are possible using UNI 4.0 signalling. As such, a network technology like ATM is especially feasible in the context.

- **Network layer protocol support**

The possibilities of network layer protocol support for layer-encoded media streams can be seen by conducting a comparison between the original RSVP approach and MMG approach [Zhang93, Zhang94]:

RSVP is a new resource reservation protocol proposed by the Internet community. In the initial design, the RSVP protocol sets up reservation state which decides how to forward each packet and uses packet filter to sort packets into different classes (separate substreams) for different treatments. Therefore, it is possible to use RSVP to treat the substreams of a media stream differently and conduct media scaling in case of need.

The MMG (multiple multicast groups) approach, in contrast, uses multiple multicast groups for a layered-encoded media stream, one group for each substream. The advantages of MMG are two fold. First, it provides a convenient constraint of bandwidth selection. For example, bandwidth requirements can be made in relationship to substreams and receivers can choose their bandwidth usage be deciding which multicast group(s) to join. Second, it provides a finer control granularity for multicast routing. Since routing decides where to forward packets, the MMG method can make substreams visible to QoS routing, so that substreams can be routed through small pipes that the whole stream cannot. In case of necessity, unwanted substreams can be pruned (truncated).

By the way, in the newest version of RSVP draft, the idea of MMG is also considered to be a better choice and only a very restricted set of filters is allowed.

- **Transport layer protocol support**

Transport layer protocol can support the realization and application of the FAST model by providing mechanisms for end-to-end QoS negotiation and renegotiation, and by providing mechanisms for QoS monitoring. We first look at the example of RTP QoS Monitoring and then discuss our implementation experiences with XTP-lite / ST-II more thoroughly.

In RTP [Schulzrinne92, Schulzrinne94], there are mechanisms for supporting QoS monitoring:

The packets of RTCP, the control protocol of RTP, contain the necessary information for quality-of-service monitoring and controlling. As data RTP packets, they are multicast so that all session members can survey how the other participants are functioning. Applications that have recently sent data, generates a sender report. It contains information useful for intermedia synchronization as well as cumulative counters for packets and bytes sent. These allows the receivers to estimate the actual data rate. Receiver reports are issued by all session members for all senders they have heard from recently. They contain information on the highest sequence number received, the number of packets lost, a measure of the interarrival jitter and timestamps needed to compute an estimate of the round-trip delay between sender and the receiver issuing the report. Loss and jitter information contained in the receiver reports can be used by senders to adjust their sending rate thus achieving graceful degradation in case of need.

The BERKOM-II MMT protocol stack XTP-lite / ST2, as we implemented on a SUN platform [Fan95f], possesses the necessary mechanisms for QoS negotiation and renegotiation, and the support for QoS realization and monitoring. Some concrete cases are illustrated in the following.

The XTP-lite / ST2 transport service provides mechanisms for the definition of the users requirements in terms of QoS parameters. A small set of clearly defined QoS parameters are included: TSDU maximum size (bytes), throughput (TSDUs / second), end-to-end transit delay (miniseconds), and reliability class (4 classes available). Rate control mechanisms and timing control mechanisms, for example, are used to guarantee the negotiated throughput.

In our implementation, QoS negotiation can be conducted at the time of connection establishment in a form of “handshaking”. If necessary, the transport user can require the transport entity to initiate a new round of QoS negotiation at any time. In terms of an extended socket API, this is done by calling the following function:

```
setsockopt(so, XTP_lite, B2_SET_QOS, &new_qos, sizeof(new_qos))
```

In the current version of our implementation, a transport responder can either acknowledge the new QoS request positively by accepting it, or tentatively reduce the QoS parameters in its acknowledgment, or acknowledge the new QoS request negatively by rejecting it. The responder is not allowed to raise the QoS parameters. On receipt of the acknowledgment with the possibly reduced QoS parameters from the transport responder, the initiator can either accept it implicitly or reject it explicitly. In this way, an endless negotiation circle will not occur.

The transport applications can also be informed of QoS violation conditions. For example, the violation of QoS at receiving side is indicated to the transport application by way of the setting of a flag `B2_QOS_VIOLATION` in the extended `recv` system call. The application then needs to check the condition in the following way:

```
if (flag & B2_QOS_VIOLATION) {
    /* Reaction handling to QoS violation, such as new QoS
        negotiation. */
}
```

With the transport services similar to these and others provided by our XTP-lite/ST-II implementation, it should be feasible to construct an adaptive communication application which can handle application-initiated adaptation as well as handle supporting-system-initiated adaptation.

### 9.3.4 Adaptation support from OS scheduler

Without the architectural support for protocol processing and the scheduling mechanism support for all system activities, the FAST model can only be realized and applied in a primitive and approximate way. This became quite evident for us as we tried to implement some functions of XTP-lite/ST-II processing according to the FAST model on a SUN platform, where time-sharing scheduling, primitive timing support and BSD protocol processing architecture are the main environmental conditions [Fan95f].

We have discussed the issue of realizing predictable protocol processing architecture by application-driven protocol processing in Chapter 8. To make an adaptation to changing environments really effective, the related processes and their timing constraints must also be adapted. Above all, this requires the support from the OS scheduler.

By using a soft real-time scheduling framework, it is easier to accommodate the changing environments. This is mainly due to two features of the soft real-time framework: (1) imprecise timing-constraint descriptions can be tolerated to some extent; (2) admission control can be made quickly and in an optimistic manner. Because of the two features, the small fluctuations in the execution timing of the processes can be tolerated, for example, in a framework like CO-SCHEDULE or SIM-SCHEDULE (see Chapter 5 and Chapter 7). The following will deal with the major changes in the execution modes.

- **Timing property changes initiated by the user-processes**

In the case of application-initiated adaptation, i. e., when the application has to change its working modes, it usually has to initiate the changes of the timing properties of some of its processes itself. It is our belief that a supporting-system-initiated adaptation should also be consented by the application processes before going into effect. For this to work, the timing-constraint changes of the processes can always be invoked by the application processes themselves actively by using corresponding programming interfaces. We do not, therefore, provide implicit, automatic methods to adapt the processes dynamically by the scheduler itself. The same policy usually applies to other management subsystem as well. This simplifies the implementation of the scheduling subsystem and the management of other subsystem in dealing with the adaptation dynamics. As a side effect, the possible problem of frequent oscillation caused by an automatic adaptation can also be avoided to a large extent.

One primitive way to deal with process-property changes is to cancel the current processes and recreate them with their new timing constraints. This is feasible to some extent, since the creation and deletion of processes are quite easy in such scheduling framework as CO-SCHEDULE and SIM-SCHEDULE, where simple admission control and management are used. But it is more convenient for the application to accommodate its changing modes by changing the timing control of the running processes directly. It is also less time-consuming to provide a means to adapt the processes to new timing properties without having to destroy the current processes and then to recreate them with the changed timing constraints.

In the scope of the experimental implementations of the CO-SCHEDULE and SIM-SCHEDULE scheduling subsystems, which has been described in Chapter 7 to a large extent, some possible implementation forms for process adaptations have also been experimented.

Recall from Chapter 7, the interfaces for creating a semi-imperative periodic real-time process and a virtual periodic real-time process take the following forms:

```
ProcID = CreateProc_PERIODIC (env, unit_prog, end_prog,
                              period, unit_comp_time, tv_excep)
ProcID = CreateProc_VIRTUAL (env, body_prog, end_prog,
                              period, unit_comp_time)
```

After being created, a real-time process should be explicitly started. It is at the time of trying to start a process that the system makes an admission control. An already successfully created periodic process or virtual periodic process can be started using the following:

```
success = StartProc_RT (start_mode, ProcID, start_time, end_time)
```

In our experimental implementation, an adaptation of the timing properties of a real-time process is conducted in a way of restarting the process. The scheduling subsystem has to make an admission control test again in order to see whether it is possible to let the process change its timing requirements. A real-time process can request to change its timing constraints with the following call:

```
success = AdaptProc_RT (start_mode, ProcID, new_start_time,
                        new_period, new_unit_comp_time)
```

where, *new_start_time* is the intended earliest possible time of mode change for the target (virtual) periodic process. *new_period* and *new_unit_comp_time* are the new intended period and computation time per period for the target (virtual) periodic process. As in the case of starting a real-time process, different *start_mode*'s can be used for different requirements concerning guarantee and timing wishes in adapting the process.

In order to check whether the adaptation requirement is allowed, an admission control is conducted as if, to the time of mode change, the current process with the current timing properties would be deleted and a new process with the new intended timing properties would be created. Other aspects of the admission control for adapting the process are identical to the admission control for starting the process, which is already detailed in *Section 7.3.2 Simple admission control*. For sake of simplicity, only the mixed-placement algorithm is used in the adaptations.

The timing adaptations generally fall into two categories: (1) Capacity reduction. In this category, the processing capacity required by the adaptation has either remained constant or been reduced. Then the process can usually be scheduled according to the new timing constraints as soon as an old period has come to an end and the time *new_start_time* has been reached. (2) Capacity enhancement. In this category, processing capacity required by the adaptation has been raised. As mentioned, the scheduling subsystem has to check the new CPU capacity requirements as if the current process would be deleted and a new process would be created. The admission control might not allow this change of timing-requirements.

If the timing adaptation can not pass the admission control, the *AdaptProc_RT* call returns a value of *FAILURE* and nothing is changed — the target process remains the same. If the timing adaptation can pass the admission control, the *AdaptProc_RT* call returns a value of *SUCCESS.* In the state field *ProcFlags* of the process control block (PCB), a state bit “mode change pending” is set and the values of *new_start_time, new_period, new_unit_comp_time* are also recorded.

At the start of each new period, a process with its “mode change pending” bit set will check whether it has reached its *new_start_time*. If yes, then the values of *period, unit_comp_time* in PCB are replaced with *new_period, new_unit_comp_time* and the “mode change pending” bit is cleared. (These extensions are made in the “Reinstall TimerJob” part of the flow chart Figure 7-10 on page 183). The process is then scheduled according to the modified timing properties automatically.

The sketched implementation scheme is apparently more efficient than the primitive approach of deleting the existing process and recreating a new process.

- **Monitoring and bottom-up notification of overload**

The SIM-SCHEDULE scheduler monitors the execution of the high-priority periodic processes. The scheduler preempts a high-priority sure-/maybe- guaranteed periodic process and possibly renders it into a process of normal priority class m if this periodic process has not completed its task in one of its period after using up its claimed execution time. (See, for example, *Section 7.3.4 Implementation of soft real-time handling*.) The execution control of the process can then be transferred to a timing-violation handler defined by this process. That is, the timing-violation handler defined by this process will be invoked as soon as the process is allowed to execute again. Under the control of the SIM-SCHEDULE scheduler, the mal-function of one of the high-priority periodic processes will not have negative effects on other high-priority periodic processes and the processes of the priority class 1 to (m-1).

By timing-violation handler, the scheduling subsystem notify an application process that it is unlikely to schedule its activity according to its time-constraints. This mechanism allows the application processes to react to the missed deadlines in an application-specific manner. Such a notification mechanism is one of the key features with which a supporting-system-initiated adaptation can be built.

## 9.3.5 Support from other OS resource pools

In addition to CPU processing power administered by an OS scheduler, various other OS resource pools should be used to realize resource reservation, soft guarantee, monitoring, and adaptation. Some of the related investigations have been presented in the preceding chapters and will not be repeated here. Still some other considerations are necessary. For example, it is necessary to have high-resolution timing facilities in order to conduct monitoring and adapting precisely.

For communication applications, access to buffer pool is necessary. A partition of the whole buffer pool is then necessary: statically reserved partitions can be reserved to processes requiring guarantees. Other access requirements can be met by sharing some common buffer partition.

### 9.3.6 Automatic higher-level adaptation management

In a composite application scenario, some higher-level adaptation managements can be done in an automatic or semi-automatic manner. A set of user-defined preferences and policies can be set up before hand to assist the automatic or semi-automatic choice of which sub-applications to adapt in case of need. The run-time decisions concerning adaptations can then be done on the basis of such a set of preferences or policies.

The human ears are usually more sensible to the changes in audio quality, whereas the eyes can better tolerate the instability of video quality for a while. In a multimedia collaboration (MMC) application, therefore, the user might always prefer the degradation of video quality to the degradation of audio quality, if a degradation is unavoidable. Such preferences can be applied to the MMC software so that the video-related processes are always requested to lower their processing capacities if an overload condition is detected. Another example is the resource allocation difference between different working modes. The user can predefine a “normal working mode” in which all activities are allocated some portions of resources to go ahead, a “mute mode” in which the local site will not send out any A/V streams, a “local mode” in which A/V interaction with other sites are minimized to let the local activities such as compiling to go as fast as possible, a “document exchange mode” in which A/V interaction with other sites are minimized whereas ftp activities with other sites are given the most favor, etc.

## 9.4 Summary of the chapter

The algorithms and techniques proposed in the preceding chapters have laid a foundation for building soft real-time supporting systems that are quite predictable under general and normal operations. However, macro-level application-initiated adaptations and the supporting-system-initiated adaptations are still needed by many multimedia applications under certain circumstances. Our adaptive service model aims primarily at serving these macro-level adaptations, which are still directly supported and predictively controlled by the supporting system.

The service model tries to provide services in a flexible and adaptive manner (called the FAST model). By the FAST model, the construction of the (supporting) system includes not only normal processing components but also management component, monitoring component and adjustment component. Two aspects make it possible to use the FAST adaptive service model: the adaptive nature of some multimedia applications and the technical bases that supports adaptability. The latter includes media coding and media scaling, communication protocol mechanisms and other OS supports.

We argue that flexible real-time OS supports are vital to make an adaptive system support complete. Soft real-time techniques are feasible for this task since they can not only provide different degrees of soft guarantee, which make the system basically predictable, but can also accommodate the flexibility of adapting to changing system environments. Some proposals for providing such OS supports have been made and experimented in our experimental implementations.

# PART V

# Conclusion

# Chapter 10

# Conclusions and future work

We now summarize the main contributions of the dissertation, compare them with other related work and point out possible working areas for future research.

## 10.1 Summary of main contributions and comparisons with related work

Throughout the dissertation, comparisons have been made between our work and other related work, whenever possible and appropriate. Here once again, we highlight our main contributions [Fan94a, Fan95a, Fan95d, Fan96b] and emphasize the main differences between our contributions and those of related work.

Our central idea and starting point for supporting multimedia (especially continuous media) communications and applications is to exploit the special characteristics of them. That is, the support of OS to continuous media should exploit the special features of continuous media which are not present in or are not typical of other kinds of computer and communication applications. Some of these features can be identified as soft real-time, soft guarantee, periodicity, error-tolerance, adaptability, etc.

Based on a survey and analysis of the state-of-the-art of operating system support for multimedia, this dissertation has made its main investigations on the following topics:

### 10.1.1 On a soft real-time framework

Many researchers have pointed out and experimented with the possibility of applying real-time technologies to multimedia systems [Anderson90a, Herrtwich92, Jeffay92, Leslie93, Mercer94]. Most of them have applied the real-time mechanisms and algorithms to protocol processing or other communication-related activities only. The situation remains the same with some recent work [Coulson95, Wolf96, Gopal96, Mehra96]. As argued in Part I, this is neither enough to control the whole multimedia system in a predictable manner nor feasible for a more complicated system environment like multimedia collaboration.

As a solution, we propose to use a soft real-time framework to control all the activities on an end-system. As a system framework, a soft real-time framework will inevitably involve both hardware and software components, with the operating system as its center. For the part of OS, our soft real-time framework proposal consists of a process framework for categorizing processes, some timing enforcement models and some base soft real-time scheduling and handling schemes.

The work by [Mercer94a] has a similar proposal of controlling activities of a multimedia system under a real-time framework. Their motivation is to provide “a unifying framework for hard real-time and multimedia systems”. Therefore, they advocate the direct application of hard real-time theory to support multimedia applications. Our work has furthered their ideas. Our soft real-time framework just tries to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system can be run in a more or less predictable manner. By exploiting the soft real-time properties of the continuous media applications, this approximation can usually be realized in simple and efficient ways.

It is also important to point out that the process categorization criteria of our framework are not directly multimedia-dependent. In our framework, the guarantees are not only accessible by CM-related activities. Other non-CM-related activities can also receive guarantees if they are needed. This feature of our framework is especially important for composite MMC scenarios. The frameworks by other work have quite often some limitations on this aspect.

As to the realization of the soft real-time framework, we argue that it is not necessary to go so far as to use a vertically integrated nano-kernel approach as used in [Roscoe95, Leslie96], we advocate a more traditional micro-monolithic approach.

### 10.1.2 On soft real-time scheduling and handling schemes

Soft real-time scheduling and handling schemes are essential for the realization of a soft real-time framework. A set of scheduling and handling schemes have been designed which can be used in a concrete multimedia system. The schemes are simple, predictable, flexible yet powerful. Extensive simulations and implementations have been conducted to validate their feasibility.

The Generalized Rate Monotonic Theory (GRMT) has been chosen as the theoretical basis for the design of our soft real-time scheduling and handling schemes. Our scheduling schemes are designed by integrating some elements from the rate-monotonic scheduling, the priority-based scheduling and the weighted round-robin scheduling. The Generalized Rate Monotonic Theory is used in all the methods as a basis for admission control and for real-time scheduling.

For soft real-time systems, not-quite-often timing overflows caused either by the system or by the application processes are expected phenomena. Their handling by soft real-time handling schemes is a normal part of the system operation. Our soft real-time handling schemes are mostly corrective in the sense that they try to smooth system fluctuations.

Compared to other related work [Nieh93, Wolf96], our schemes are based on a sound real-time theory GRMT and use corresponding admission control to avoid unexpected system saturations. Direct supports in the forms of periodic real-time processes and virtual periodic real-time processes are not only provided to MM-relevant activities but also provided to other non-MM-relevant activities, which should receive guaranteed processing bandwidth to go forward regularly in case of need. Through timing handling schemes, timing overflows can also be handled in graceful manners.

### 10.1.3 On predictable protocol processing and adaptation support

In order to contain the communication-related activities in the soft real-time framework, protocol processing must be made predictable. Methods have been investigated which can be used to reduce the influences and randomness of the interrupt processing caused by incoming packets. They either exploit the characteristics of continuous media communication to make interrupts less and regular, or, simply make use of the possibility of physical reservation and separation. Two-level interrupt processing is also advocated where the first-level packet interrupt processing is made brief while the main protocol processing is conducted in the second level and is scheduled in the framework of soft real-time processing. Processing capacity for the common-path interrupt processing is reserved in the form of some virtual periodic system-overhead processes and the main protocol processing is conducted and reserved in an application-driven manner.

The ideas of putting part of the lower layer protocol processing in user-space have been used in many other work [Thekkath93, Maeda93, Edwards94]. Their emphases are mainly on the flexibility and efficiency in implementing the protocol processing. Our emphasis, in contrast, is put on the predictability. That is, user-layer protocol processing make the reservation of CPU capacity for protocol processing more application-oriented, easy and precise.

Although the algorithms and techniques of the soft real-time framework have laid the foundation for building soft real-time supporting systems that are quite predictable under general and normal operations, some macro-level application-initiated adaptations and supporting-system-initiated adaptations are still needed by many multimedia systems under certain circumstances. In addition to other supports such as media scaling and communication protocol mechanisms, we argue that flexible real-time OS supports are vital to make an adaptive system support complete. Soft real-time techniques are feasible for this task since they can not only provide different degrees of soft guarantee, which make the system basically predictable, but can also accommodate the flexibility of adapting to changing system environments. Some proposals for providing such OS supports are made and are experimented in our scheduling subsystem.

It is clear that we do not promote a system that relies exclusively on "graceful degradation" to attempt to handle system overload, as is done in [Compton94, Fall95]. Our adaptation supports are still predictively controlled by the soft real-time framework.

### 10.1.4 On theoretical analyses, simulations and implementations

Operating systems that support multimedia applications must modify the traditional view of scheduling and resource allocation in order to accommodate the need for timeliness and to deal with the constraint of limited resource. A new set of basic concepts, principles and algorithms on resource management and scheduling has been developed. They deal with components of a soft real-time framework, semantics of soft guarantee, real-time process model, criteria of suitable scheduling schemes, soft real-time attributes, timing enforcement model, algorithms for scheduling and handling, admission control etc. As a theoretical exploration, mathematical bounds of soft real-time with loss is considered under different assumptions about the deadline and computation time distributions of the arriving processes. They are modeled by means of queueing models and their theoretical performance limits are derived.

Simulations and implementations are conducted to further ascertain the feasibility of the soft real-time framework and to evaluate the related soft real-time scheduling and handling methods. The simulations have evaluated and compared the proposed schemes under a wide range of system parameter settings. Some of the quantitative evaluation results from the simulations have been used in choosing algorithms and their parameters in the prototype implementations. The experimental implementations and related measurements have further verified the feasibility of our approach and algorithms. The practical experiences such as making the scheduler as a timing enforcer and timing predictor in one are quite enlightening for more practical work in this direction.

## 10.2 Directions for future research

In retrospect, the goals set up at the beginning of the dissertation have been reached. Our work has developed a set of models, methods and algorithms suitable for multimedia operating system support. As a whole, they constitute a novel architecture which provides general and efficient support for multimedia. The basic ideas of them can also be separately applied to the existing OS environments so that the usability or the performance of the existing systems can be improved to support the multimedia applications better.

Based on the current work, research and development can be continued in many ways. Refinements and evaluations of some variations of the soft real-time scheduling and handling schemes may be conducted; more specific strategies for the scheduling of other resources other than timing scheduling may be explored; a kind of service definition with quantitative matrix may be specified so that many kinds of QoS can be managed in a management information base (MIB); ......; it is most desirable, of course, if the proposed models and algorithms can be used in a complete system solution with large-scale practical application scenarios.

The work on multimedia support is by far not complete yet. There are still many challenging problems to be tackled. Through the research and development on OS support for multimedia, we believe, a more solid basis can be created on which more efficient and powerful multimedia systems can be built.

# Appendix: ET-SCHEDULE

in Chapter 5, an innovative soft real-time scheduling scheme named ET-SCHEDULE has been mentioned. In this appendix, the scheduling scheme will be described in more detail. ET-SCHEDULE stands for an **e**lastic **t**ime-scale soft real-time scheduling scheme (or drifting clock scheme). The scheduling scheme is aimed at avoiding some disadvantages brought about by the traditional scheduling schemes including CO-SCHEDULE and SIM-SCHEDULE.

- **Problems and questions concerning timing-violation handler method**

The motivation for introducing an elastic time-scale soft real-time scheduling scheme is that many questions can sometimes be raised to a timing-violation handler method based on a rigid time scale:

How should the programmers of the real-time (RT) system take timing violation/emergency conditions into explicit consideration? How to express and program it?

At what priorities should these emergency handlers be scheduled?

To what extent should/could the handlers change the normal behavior of their parent RT process, for example, by changing their RT timing attributes or even aborting their further processing?

Which one causes more disturbance to the whole system: to activate and execute an timing-violation handler, or, to let the extra part of the user process, which exceeds the proclaimed deadline, simply run to end?

In addition, the currently running process should sometimes produce intermediate inputs which are necessary for the processes followed. Can the processes followed still make reasonable progress even though the current process does not produce all the intermediate results because its time-budget is used up?

- **Elastic time scale scheduling model**

The elastic time scale scheduling model frees the RT processes of the above burdens by providing them with the following features: the RT processes will never have timing violations thus they do not need any emergency handler. The programming model for these RT processes are quite simple: they always assume their timing characteristics are guaranteed by the scheduling system.

The idea of the drifting clock is to slow down or to speed up logical time in case of need: slow down time for the real-time tasks which have a longer actual running time than they previously estimated (and claimed) through their real-time parameters so that they can get their work done. And, speed up the time when the system is idle or is executing non-real-time tasks.

- **Operational scheme**

There are several forms of elastic time scale. In all these elastic time scale schemes, two time scales are used as time references — a physical time scale which is produced by the exact physical clock and a drifted logical time scale which is produced by a drifting logical clock.

A simple form is explained first. In this first form of elastic time scale, logical time is always equal to or slower than physical time.

Scheduling is done according to the logical time scale. The logical clock usually runs at the same pace as the physical clock or it sometimes runs behind the physical clock. A real-time task is always executed till the end of its period once it is started. At the end of each period of the real-time task execution, the logical clock is advanced / adjusted according to the claimed logical execution time of the real-time task instead of according to the real physical execution time of the task. But the logical clock is never adjusted ahead of the physical clock. (The physical clock runs, of course, always at its physical pace.) The logical clock might be behind the physical clock because real-time tasks might have run physically longer than they formally (logically) claimed. If the logical clock runs behind the physical clock, then it should try to run at a faster rate in order to catch up with the physical clock.

Figure 10-1 shows a section of the scheduling scenario. In the figure, $r_i$'s are real-time (RT) tasks and $t_i$'s are non-real-time (non-RT) tasks. It is assumed that all the tasks claim logically an estimated execution time of 4 time units. It is further assumed that the logical clock runs two times as fast as the physical clock in its “faster” mode when the logical time is behind the physical time. At logical time ($T_L$) and physical time ($T_P$) 0 ($T_L = T_P = 0$), RT task $r_1$ is started and it takes 4 physical time units as claimed. Both the physical clock and the logical clock are advanced to time 4. The following RT task $r_2$ has actually run for 6 physical time units instead of the logically claimed 4 time units. At the end of $r_2$ period, the physical clock is advanced to 10, but the logical clock is only advanced to 8 according to the claimed logical execution time of $r_2$. Now the logical clock is behind the physical clock and it runs now two times as fast as the physical clock. The non-RT task $t_3$ takes then 4 logical time units but actually only 2 physical time units because the physical clock runs now faster. At $T_L$=$T_P$=12, the two clocks run at the same pace again. RT tasks $r_{3,}$ $r_5$ and $r_6$ take less than 4 physical time units, but the logical clock is at maximum adjusted to be the same as the physical clock, not faster than the physical clock.

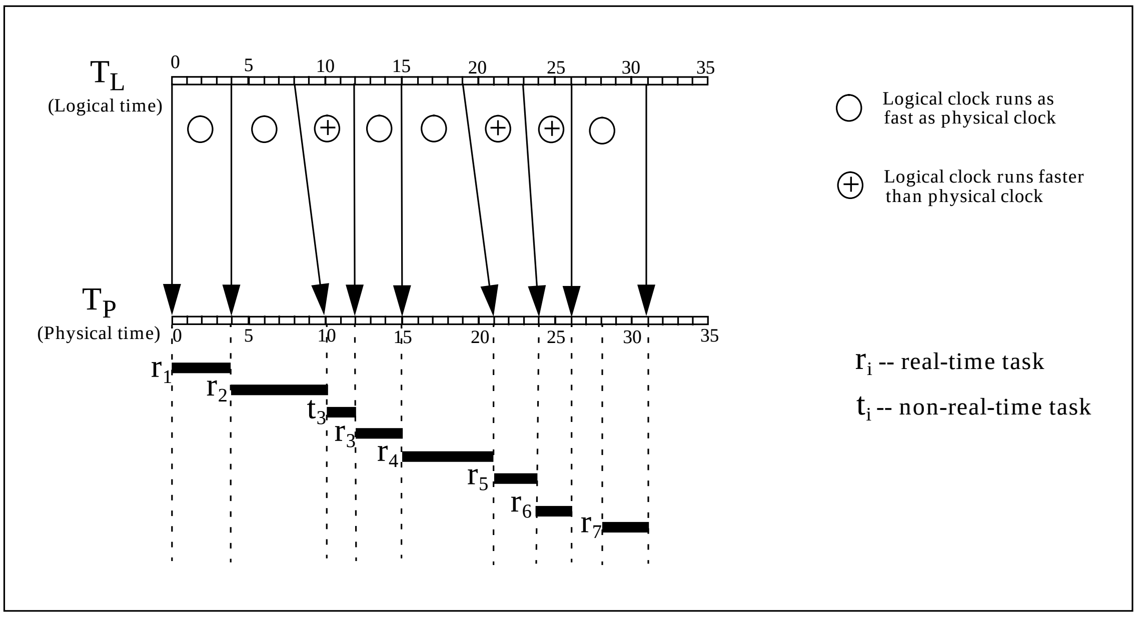

***Figure 10-1*** **Drifted logical time vs. actual physical time**

- **Rationales and variations**

The rationales for such an elastic time scale are:

1) Do not use rigid deadline — the value function of a process does not usually

become zero after the deadline;

2) Deadline violation need not lead to some kinds of exception handling, or even worse, to the abortion of the processes;

3) Compensations between the worst-case-execution time of different processes are usually possible.

The above simple form of elastic time scale can be extended to several variations.

Variation 1 — With Elasticity Boundary (EB). The drifted logical time can be at maximum EB time units slower than the exact physical time. If the drifted logical time is EB behind the exact physical time, a catch-up in the logical time is done in that the logical time is assigned the value of the physical time. (All the tasks which should have been executed in the “catch-up gap” are then ignored.)

Variation 2 — With violation handler. To invoke timing-violation handler when a RT task makes the logical clock drift two much.

Variation 3 — Free drifting. The logical time can be slower or faster than the physical time.

Note that, although non-real-time processes usually still get their shares to go forward, their physical shares are actually getting smaller when the logical clock runs faster than the physical clock. In this way the real-time processes are favored in a new aspect. In addition, for some forms of elastic time, some adjustment points for the logical clock may be introduced: when there are no ready RT tasks to run, then the logical time and physical time are adjusted to be the same at once. This method has the effect that all the non-RT tasks which should have been executed in the “catch-up gap” are ignored.

- **System effects of elastic time scale scheduling**

Elastic time is one of the methods of dealing with the problem of estimating the execution time of a process. With this method or other compensation method such as timing-violation handler, the worst execution time of a process need not be estimated so pessimistically that only a low utilization can be achieved. The biggest advantage of the elastic time scheme is that the RT users of the process scheduling system have now a very simple view of the system: they can program their processing in such a way as if all the processing will be completed as they expect, i.e., no deadline violation, no exception handling, and they can have a static flow control of their own.

Note that such virtual-time based RT scheduling is only feasible in the context of soft real-time scheduling for the reason that a strict adherence to the real world time-scale is not necessary for some soft real-time environments. It is not feasible for hard real-time scheduling. Actually, elastic time scheme will not cause a mess only if (a) the applications have a certain degree of elasticity in viewing the time scale, (b) the durations of time deformation from physical time to logical time are usually short, and (c) the frequencies of such deformation are low. This is the case for some MM applications with soft-real time requirements.

Elastic time scale provides a framework in which many kinds of the existing RT scheduling methods can be adapted to function. Note also that it is possible to schedule some of the real-time processes in a system using the elastic time scale scheduling method while scheduling others using the traditional scheduling methods.

# References

| | |
|---|---|
| **[Accetta86]** | M. J. Accetta, et al, “Mach: A New Kernel Foundation for UNIX Development,” *Proc. USENIX Summer’86,* July, 1986. |
| **[Adam94]** | J. F. Adam, H. H. Houh, M. Ismert & D. L. Tennenhouse, “A Network Architecture for Distributed Multimedia Systems,” *Proc. International Conference on Multimedia Computing and Systems*, May 1994. |
| **[Altenhofen93]** | M. Altenhofen, J. Dittrich, et al, “The BERKOM Multimedia Collaboration Service,” *Proc. ACM Multimedia93*, 1993. |
| **[Anderson90a]** | David P. Anderson, “Meta-Scheduling for Continuous Media,” *Technical report No. UCB/CSD/90/599*, UC Berkeley, Oct. 1990. |
| **[Anderson90b]** | David P. Anderson, et al, “Support for Continuous Media in the DASH System,” *Proc. 10th Int’l Conf. on Distributed Computing Systems*, IEEE, 1990. |
| **[Anderson91]** | D. P. Anderson, L. Delgrossi & R. G. Herrtwich, “Process Structure and Scheduling in Real-Time Protocol Implementations,” *Proc. GI/ITG Fachtagung Kommunikation in verteilten Systemen*, Springer, Feb. 1991. |
| **[Armand91]** | Francois Armand, “Give a Process to Your Drivers,” *Proc. of the EuroOpen Autumn 1991 Conference*, Budapest, Hungary, Sept. 1991. |
| **[Audsley91]** | N. Audsley & A. Burns, “Real-Time System Scheduling,” *Report YCS134*, Dept. of Computer Science, Univ. of York, 1991. |
| **[Baccelli84]** | F. Baccelli, P. Boyer & G. Hebuterne, “Single-Server Queues with Impatient Customers,” *Advances in Applied Probability, Vol. 16,* 1984 |
| **[Barnerjea94]** | Anindo Banerjea, Domenico Ferrari, et al, “The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences,” *TR-94-059*, ICSI & UC Berkeley, Nov. 1994. |
| **[Blair92a]** | G. S. Blair, et al, “A Network Interface Unit to Support Continuous Media,” *Technical Report MPG-92-19*, Distributed Multimedia Research Group, Lancaster University, 1992. |

**[Blair92b]** G. S. Blair, et al, “The Role of Operating Systems in Object-Oriented Distributed Multimedia Platforms,” *Technical Report MPG-92-51*, Distributed Multimedia Research Group, Lancaster University, 1992.

**[Braden96]** R. Braden, L. Zhang, et al, “Resource ReSerVation Protocol (RSVP) — Version 1 Functional Specification,” *Internet Draft draft-ietf-rsvp-spec-12.txt*, IETF, Apr. 1996.

**[Campell96]** A. Campell, C. Aurrecoechea & L. Hauw, “A Review of QoS Architectures,” *Proc. 4th IFIP Int’l Workshop on Quality of Service*, Paris, Mar. 1996.

**[Casavant88]** T. Casavant & J. G. Kuhl, “A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems,” *IEEE Trans. on Software Engineering*, Vol. 14, No. 2, Feb. 1988.

**[Casner92]** S. Casner & S. Deering, “First IETF Internet Audiocast,” *Computer Communications Review*, Vol. 22, No. 3, ACM, July 1992.

**[Cheng88]** S.-C. Cheng & J. A. Stankovic, “Scheduling Algorithms for Hard Real Time Systems — A Brief Survey,” in *Hard Real Time Systems*, J. A. Stankovic & K. Ramamritham, eds., IEEE, 1988.

**[Chung90]** J.-Y. Chung, J. W. S. Liu & K.-J. Lin, “Scheduling Periodic Jobs that Allow Imprecise Results,” *IEEE Trans. on Computers, Vol. 39, No. 9,* Sept. 1990.

**[Clark90]** David D. Clark & David L. Tennenhouse, “Architectural Considerations for a New Generation of Protocols,” *Proc. SIGCOMM’90*, ACM, 1990.

**[Clark92]** D. Clark, S. Shenker & L. Zhang, “Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism,” *Proc. SIGCOMM’92*, ACM, 1992.

**[Colnaric90]** M. Colnaric, “Run-Time Prediction for Hard Real-Time Programs,” *PEARL90 - Workshop über Realzeitsystems*, Springer, 1990.

**[Compton94]** C. Compton & D. L. Tennenhouse, “Collaborative Load Shedding for Media-Based Applications,” *Proc. International Conference on Multimedia Computing and Systems*, May 1994.

**[Coulson93a]** G. Coulson & G. S. Blair, “Micro-Kernel Support for Continuous Media in Distributed Systems,” *Technical Report MPG-93-04*, Department of Computing, Lancaster University, 1993.

**[Coulson93b]** G. Coulson, G. S. Blair, P. Robin & D. Shepherd, “Extending the

Chorus Micro-Kernel to Support Continuous Media Applications," *Proc. 4th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video,* Nov. 1993.

**[Coulson95]** G. Coulson & G. Blair, "Architectural Principles and Techniques for Distributed Multimedia Application Support in Operating Systems," *ACM Operating Systems Review,* Oct. 1995.

**[Cramer92]** A. Cramer, M. Farber, B. McKellar & R. Steinmetz, "Experiences with the Heidelberg Multimedia Communication System — Multicast, Rate Enforcement and Performance," *Proc. High Performance Networking'92*. IFIP, 1992.

**[Danthine92]** A. Danthine, et al, "The OSI 95 Connection-Mode Transport Service — The Enhanced QoS," *Proc. High Performance Networking'92*, IFIP, 1992.

**[Deering95]** S. Deering, "Reservations or No Reservations," *INFOCOM'95 panel presentation slides,* Apr. 1995.

**[Delgrossi93a]** L. Delgrossi, et al, "Media Scaling in a Multimedia Communication System," *Proc. 1st ACM Multimedia Conference,* 1993.

**[Delgrossi93b]** L. Delgrossi, R. G. Herrtwich, C. Vogt & L. C. Wolf, "Reservation Protocols for Internetworks: A Comparison of ST-II and RSVP," *Proc. 4th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video,* Nov. 1993.

**[Dittrich95]** J. Dittrich, et al, *Multimedia Collaboration User's Manual for SunOS (MMCS Release 2.7)*, GMD-FOKUS, Mar. 1995.

**[Dupuy92]** S. Dupuy, W. Tawbi & E. Horlait, "Protocols for High-Speed Multimedia Communications Networks," *Computer Communications*, Vol. 15, No. 6, July 1992.

**[Edwards94]** A. Edwards, et al, "User-Space Protocols Deliver High Performance to Applications on a Low-Cost Gb/s LAN," *Computer Communication Review,* ACM SIGCOMM, Sept. 1994.

**[Fall95]** K. Fall, J. Pasquale & S. McCanne, "Workstation Video Playback Performance with Competitive Process Load," *Proc. 5th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'95)*, 1995.

**[Fan92]** Changpeng Fan, "Performance Limits of Time-Constrained Message Transmission over a Multiple Access Channel," *Technical Report FB20-1992-3,* Dept. of Computer Science, Technical University of Berlin, Feb. 1992.

**[Fan93]** C. Fan, T. Luckenbach & X. Xu, "Performance Comparison and

Analysis of XTP and TCP/IP over the BERKOM Broadband ISDN Network," *Proc. IEEE 12th Annual Conf. on Computer Communications (INFOCOM'93)*, IEEE, 1993.

**[Fan94a]** Changpeng Fan, "MMOSS: Soft Real-Time Operating System Support in a Multimedia Communication Subsystem," *Proc. 19th IFAC/IFIP Workshop on Real-Time Programming*, Konstanz, Germany, IFAC, June, 1994.

**[Fan94b]** Changpeng Fan, "Continuous Media Communication and Real-Time Scheduling," in *1. Meilensteinbericht für das Kooperationsprojekt "Spezifikation eines multimediafähigen Transportsystems für ATM-Netze" zwischen Siemens AG und TU Berlin-OKS*, Mar. 1994.

**[Fan95a]** Changpeng Fan, "Realizing a Soft Real-Time Framework for Supporting Distributed Multimedia Applications," *Proc. 5th IEEE Workshop on the Future Trends of Distributed Computing Systems (FTDCS'95)*, Korea, Aug. 1995.

**[Fan95b]** C. Fan & J. Selent, "Experimentelle Implementierung von Echtzeitplanungsmechanismen für Multimediale Anwendungen auf PPDS," in *5. Meilensteinbericht für das Kooperationsprojekt "Spezifikation eines multimediafähigen Transportsystems für ATM-Netze" zwischen Siemens AG und TU Berlin-OKS*, Sept. 1995.

**[Fan95c]** Changpeng Fan & R. Ruppelt, "Protocol Performance Measurements in a Heterogeneous Network Environment," *Performance Evaluation Journal, Volume 22, Issue 3*, Elsevier, May 1995.

**[Fan95d]** Changpeng Fan, "An Adaptive Service Model for Supporting Multimedia Communications and Applications," *Proc. 2nd Asia-Pacific Conference on Communications (APCC'95)*, Osaka, Japan, June 1995.

**[Fan95e]** Changpeng Fan, "Using soft real-time scheduling schemes to support continuous media," *Proc. IEEE Int'l Conf. on Multimedia Networking*, Aizu, Japan, Sept. 1995.

**[Fan95f]** C. Fan, et al, "Experiences with a Protocol Stack for Multimedia Transport," *Proc. IEEE SICON/ICIE'95*, Singapore, IEEE, 1995.

**[Fan96a]** Changpeng Fan, "Soft Real-Time Handling Methods in a Soft Real-Time Framework," *Participants' Proc. 6th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'96)*, Japan, Apr. 1996.

**[Fan96b]** Changpeng Fan, "Evaluations of Soft Real-Time Handling Algorithms for Continuous Media," *Proc. Int'l Conf. on Multimedia Modeling*, Toulouse, France, Nov. 1996.

**[Fan97]** Changpeng Fan, et al, “Experiences with the MMT/NG Architecture for Integrated Services,” GMD FOKUS, 1997.

**[Ferrari90]** D. Ferrari, “Client Requirements for Real-Time Communication Services,” *IEEE communications Magazine*, Vol. 28, No. 11, Nov. 1990.

**[Ferrari92]** D. Ferrari, A. Banerjea & H. Zhang, “Network Support for Multimedia — A Discussion of Tenet Approach,” *TR-92-072*, ICSI & UC Berkeley, Nov. 1992.

**[Ferrari95]** D. Ferrari, “Reservations or No Reservations,” *INFOCOM’95 panel presentation slides*, Apr. 1995.

**[Fiddler89]** J. Fiddler & J. Fogelin, *The VxWorks Real-Time Kernel*, Wind River Systems, Inc., 1989.

**[Fisher92]** Tom Fisher, “Real-Time Scheduling Support in Ultrix-4.2 for Multimedia Communication”, *Proc. 3rd Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, CA, Nov. 1992.

**[FOKUS92]** FOKUS, *BERMATE — Bergate mit ATM Erweiterung*, GMD-FOKUS, Aug. 1992.

**[FORE93]** Fore Systems, *ForeRunner SBA-100 SBus ATM Computer Interface — User’s Manual, Release 2.1*, Fore Systems, Inc., 1993.

**[FORE95]** Fore Systems, *Programmer’s Reference Manual for AALI Interface*, Fore Systems, Inc., July 1995.

**[Gien91]** M. Gien, “Next Generation Operating Systems Architecture,” *Operating Systems of the 90s and Beyond, LNCS 563*, Springer, 1991.

**[Gopal96]** R. Gopalakrishnan & G. M. Parulkar, “Efficient User Space Protocol Implementations with QoS Guarantees using Real-Time Upcalls,” *Technical Report WUCS-96-11*, Dept. of CS, Washington University in St. Louis, 1996.

**[Govindan91]** R. Govindan & D. P. Anderson, “Scheduling and IPC Mechanisms for Continuous Media,” *Proc. ACM Symp. on Operating Systems Principles*, ACM, Vol. 25, No. 5, Oct. 1991.

**[Hagsand94]** O. Hagsand & P. Sjödin, “Workstation Support for Real-Time Multimedia Communication,” *Proc. USENIX Winter’94*, 1994.

**[Hayter91]** M. Hayter & D. McCauley, “The Desk Area Network,” *ACM Operating Systems Review*, Oct. 1991.

**[Hehmann90]** D. B. Hehmann, M. G. Salmony & H. J. Stuettgen, “Transport

Services for Multimedia Applications on Broadband Networks," *Computer Communications*, Vol. 13, No. 4, May 1990.

**[Herrtwich92]** R. G. Herrtwich, "The Role of Performance, Scheduling, and Resource Reservation in Multimedia Systems," *Operating Systems of the 90's and Beyond, LNCS 563*, Springer, 1992.

**[Hewitt96]** K. Hewitt, "A Survey on Desktop Videoconferencing Products," available in WWW as *http://www3.ncsu.edu/dox/video/survey.html*, Feb. 1996.

**[Hildebrand93]** D. Hildebrand, "A Microkernel POSIX OS for Realtime Embedded Systems," *Proc. Embedded Computer Conference*, California, April, 1993.

**[Hopper90]** A. Hopper, "Pandora — An Experimental System for Multimedia Applications," *ACM Operating Systems Review*, Vol. 24, Apr. 1990.

**[Hutchinson91]** N. C. Hutchinson & L. L. Peterson, "The $\chi$-kernel: An Architecture for Implementing Network Protocols," *IEEE Trans. on Software Engineering*, 17(1):64-76, Jan. 1991.

**[Hutchison94]** D. Hutchison, G. Coulson, A. Campell & G. Blair, "Quality of Service Management in Distributed Systems," in *Network and Distributed Systems Management*, M. Sloman ed., Addison Wesley, 1994.

**[Jacobson95]** Van Jacobson, "The MBone — Interactive Multimedia on the Internet," *Presentation slides at UC Berkeley*, Feb. 1995.

**[Jeffay91]** Kevin Jeffay, Donald L. Stone & Daniel E. Poirier, "YARTOS — Kernel support for efficient, predictable real- time systems," *Proc. Joint IEEE Workshop on Real-Time Operating Systems and Software*, 1991.

**[Jeffay92]** K. Jeffay, D. L. Stone & F. D. Smith, "Kernel Support for Live Digital Audio and Video," *Computer Communications*, Vol. 15, No. 6, July 1992.

**[Jeffay95]** Kevin Jeffay, "A Rate-Based Execution Abstraction for Multimedia Computing," *Proc. 5th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video*, Apr. 1995.

**[Jones93]** Michael B. Jones, "Adaptive Real-Time Resource Management Supporting Modular Composition of Digital Multimedia Services," *Proc. 4th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video*, Nov. 1993.

**[Katcher93]** Daniel I. Katcher, H. Arakawa & J. K. Strosnider, "Engineering and Analysis of Fixed Priority Schedulers," *IEEE Transactions on Software Engineering, Vol. 19, No. 9,* 1993.

**[Kay93]** J. Kay & J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," *Proc. 1993 Winter USENIX*. Jan. 1993.

**[Khalidi92]** Y. Khalidi & M. Nelson, "An Implementation of UNIX on an Object-Oriented Operating System," *Spring Technical Report 92-3,* Sun Microsystems Laboratories, 1992.

**[Kihara93]** S. Kihara, et al, "An Implementation of ST-II Protocols as a User-Level Server on Real-Time Mach," *Proc. 4th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video,* Nov. 1993.

**[Klein93]** M. H. Klein, et al, *A Practitioner's Handbook for Real-Time Analysis,* Kluwer Academic Publishers, 1993.

**[Kleinrock75]** L. Kleinrock, *Queueing Systems, Vol. 1: Theory,* Wiley Interscience, 1975.

**[Kopetz89**] H. Kopetz, et al, "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach," *IEEE Micro,* Feb. 1989.

**[Kopetz91]** H. Kopetz, "Event-Triggered versus Time-Triggered Real-Time Systems," *Operating Systems of the 90s and Beyond, LNCS 563,* Springer, 1991.

**[Kurose84]** J. F. Kurose, M. Schwartz & Y. Yemini, "Multiple-Access Protocol and Time-Constrained Communication," *Computing Surveys,* ACM, Mar. 1984.

**[Kurose87]** J. F. Kurose & R. Chipalkatti, "Load Sharing in Soft Real-Time Distributed Computer Systems," *IEEE Trans. Computer, Vol. C-36, No. 8,* Aug. 1987.

**[Kurose88]** J. F. Kurose, M. Schwartz & Y. Yemini, "Controlling Window Protocols for Time-Constrained Communication in Multiple Access Networks," *IEEE Trans. Communication, Vol. 36, No. 1,* Jan. 1988.

**[Leffler89]** S. J. Leffler, M. K. McKusick, K. J. Karels & J. S. Quatermann, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, 1989.

**[Lehoczky89]** J. P. Lehoczky, L. Sha & Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. 10th Real-Time Systems Symposium,* IEEE, Dec. 1989.

**[Leslie93]** Ian M. Leslie, Derek McAuley & Sape J. Mullender, “Pegasus — Operating System Support for Distributed Multimedia Systems,” *ACM OSR*, Vol. 27, No. 1, Jan. 1993.

**[Leslie96]** Ian Leslie, et al, “The Design and Implementation of an Operating System to Support Distributed Multimedia Applications,” *Technical Report*, Computer Laboratory, University of Cambridge, Aug. 1996.

**[Liu73]** C. L. Liu & James W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *Journal of ACM*, Feb. 1973.

**[Liu91]** J. W. S. Liu, et al, “Algorithms for Scheduling Imprecise Computations,” *IEEE Computer*, 24(5):58-68, May 1991.

**[Lougher94]** P. Lougher, D. Shepherd & D. Pegler, “The Impact of Digital Audio and Video on High Speed Storage,” *Proc. 13th IEEE Symp. Mass Storage Systems*, June 1994.

**[Maeda93]** C. Maeda & B. N. Bershad, “Protocol Service Decomposition for High-Performance Networking,” *Proc. 14th ACM Symposium on OS Principles*, Dec. 1993.

**[Mathur93]** A. G. Mathur & A. Prakash, “On Transport Protocols for Audio Conferencing in CSCW Environments,” *Proc. IEEE Workshop on the Role of Real-Time in Multimedia/Interactive Computing Systems*, Nov. 1993.

**[McAuley93]** D. R. McAuley, “Operating System Support for the Desk Area Network,” *Proc. 4th Int’l Workshop on Network and Operating Systems Support for Digital Audio and Video*, Nov. 1993.

**[McCanne95]** Steven McCanne & Van Jacobson, “vic: A Flexible Framework for Packet Video,” *Proc. ACM Multimedia’95*, 1995.

**[Mehra96]** A. Mehra, A. Indiresan & K. G. Shin, “Structuring Communication Software for Quality-of-Service Guarantees,” *Proc. ACM Real-Time Systems Symposium*, Dec. 1996.

**[Mercer91]** C. W. Mercer & H. Tokuda, “An Evaluation of Priority Consistency in Protocol Architectures,” *Proc. IEEE Conf. on Local Area Networks*, Oct. 1991.

**[Mercer92]** Clifford W. Mercer, “An Introduction to Real-Time Operating Systems: Scheduling Theory,” *Technical Report*, Carnegie Mellon University, Nov. 1992.

**[Mercer94a]** Clifford W. Mercer, Stefan Savage & Hideyuki Tokuda, “Processor Capacity Reserves: Operating System Support for Multimedia Applications," *Proc. IEEE Int'l Conf. on Multimedia Computing and Systems,* May, 1994.

**[Mercer94b]** C. W. Mercer, J. Zelenka & R. Rajkumar, "On Predictable Operating System Protocol Processing," *Technical Report CMU-CS-94-165,* Carnegie Mellon University, May, 1994.

**[Milazzo91]** P. G. Milazzo, "Shared Video under Unix," *Proc. of Usenix Summer Conference,* Nashville, Tennessee, June 1991.

**[Moon95]** Sue B. Moon, Jim Kurose & Don Towsley, "Packet Audio Playout Delay Adjustment Algorithms: Performance Bounds and Algorithms," *Technical Report,* Dept. of Computer Science, University of Massachusetts at Amherst, 1995.

**[Moran92]** M. Moran & B. Wolfinger, "Design of a Continuous Media Data Transport Service and Protocol," *TR-92-019,* ICSI & UC Berkeley, Apr. 1992.

**[Mukherjee92]** B. Mukherjee, K. Schwan & P. Gopinath, "A Survey of Multiprocessor Operating System Kernels," *Technical Report GIT-CC-92/05,* College of Computing, Georgia Institute of Technology, 1992.

**[Mukherjee93]** B. Mukherjee & K. Schwan, "Survey of Real-Time Operating Systems," *Technical Report GIT-CC-93/18,* College of Computing, Georgia Institute of Technology, Mar. 1993.

**[Mullender94]** S. J. Mullender, I. M. Leslie & D. McAuley, "Operating-System Support for Distributed Multimedia," *Proc. 1994 Summer Usenix Conference,* Jun. 1994.

**[Nahrstedt95]** Klara Nahrstedt & Ralf Steinmetz, "Resource Management in Networked Multimedia Systems," *IEEE Computer,* May 1995.

**[Nakajima91]** J. Nakajima, M. Yazaki & H. Matsumoto, "Multimedia/Real-time Extensions for the Mach Operating System," *Proc. USENIX Summer'91,* 1991.

**[Nakajima92]** J. Nakajima, M. Yazaki & H. Matsumoto, "Multimedia/Real-time Extensions for Mach 3.0," *Proc. USENIX Microkernel Conference,* Apr. 1992.

**[Nakajima93]** T. Nakajima, T. Kitayama & H. Tokuda, "Experiments with Real-Time Servers in Real-Time Mach," *Proc. USENIX 3rd Mach Symposium,* 1993.

**[Nieh93]** J. Nieh, et al, "SVR4 UNIX Scheduler Unacceptable for Multimedia Applications," *Proc. 4th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video,* Nov. 1993.

| | |
|---|---|
| **[Papadopou93]** | C. Papadopoulos & G. M. Parulkar, “Experimental Evaluation of SunOS IPC and TCP/IP Protocol Implementation,” *Proc. IEEE 12th Annual Conf. on Computer Communications (INFO-COM’93)*, IEEE, 1993. |
| **[Papadopou96]** | C. Papadopoulos & G. M. Parulkar, “Retransmission-Based Error Control for Continuous Media Applications,” *Proc. 6th Int’l Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV’96)*, Japan, Apr. 1996. |
| **[Partridge91]** | C. Partridge, “Isochronous Applications Do Not Require Jitter-Controlled Networks,” *RFC 1257*, Sept. 1991. |
| **[Partridge93]** | C. Partridge, “Jacobson on TCP in 30 Instructions,” *Usenet, Comp.protocols.tcp-ip Newsgroup, Message-ID <1993Sep8.213239.28992@sics.se>*, Sept. 1993. |
| **[Partridge94]** | C. Partridge, “Gigabit Applications,” in *Gigabit Networking*, Addison-Wesley Publishing Company, Inc., 1994. |
| **[Pasieka91]** | M. Pasieka, “Distributed Multimedia: How Can the Necessary Data Rates be Supported?” *Proc. USENIX’91*, 1991. |
| **[Pawlikowski90]** | K. Pawlikowski, “Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions,” *ACM Computing Surveys*, Vol. 22, No. 2, June 1990. |
| **[Rajkumar88]** | R. Rajkumar, L. Sha & J. P. Lehoczky, “Real-Time Synchronization Protocols for Multiprocessors,” *Proc. IEEE Real-Time Systems Symposium*, IEEE, 1988. |
| **[Rajkumar90]** | R. Rajkumar, “Real-Time Synchronization Protocols for Shared Memory Multiprocessors,” *Proc. 10th Int’l Conf. on Distributed Computing Systems*, IEEE, 1990. |
| **[Ramamri89]** | K. Ramamritham, J. A. Stankovic & W. Zhao, “Distributed Scheduling of Tasks with Deadlines and Resource Requirements,” *IEEE Trans. on Computers*, Vol. 38, No. 8, Aug. 1989. |
| **[Rangan91]** | P. Venkat Rangan & Harrick M. Vin, “Designing File Systems for Digital Video and Audio,” *Proc. of the 13th ACM Symposium on Operating Systems Principles*, ACM, 1991. |
| **[Ripley89]** | G. D. Ripley, “DVI — A Digital Multimedia Technology,” *Communications of ACM*, Vol. 32, July 1989. |
| **[Roscoe95]** | T. Roscoe, *The Structure of a Multi-Service Operating System*, Ph.D Dissertation, Queens’ College, University of Cambridge, 1995. |
| **[Sandvoss94]** | J. Sandvoss, “Experiences with Media Scaling in a Multimedia |

Communication System," *Proc. 1. Arbeitstreffen der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme zur Architektur und Implementierung von Hochleistungs-Kommunikationssystemen*, GI, Jan. 1994.

**[Sasinowski95]** J. E. Sasinowski & J. K. Strosnider, "ARTIFACT: An Experimental Real-Time Window System," *Technical Report CMU-CSC-95-4,* Dept. of Electrical and Computer Engineering, CMU, 1995.

**[Schatzmayr94]** Rainer Schatzmayr & Changpeng Fan, "Providing Support for Real-Time Multimedia Data Transfer," *Proc. 4th International Conference on Communication Systems (ICCS'94),* IEEE, Singapore, Nov. 1994.

**[Schatzmayr96]** R. Schatzmayr, *Providing Support for Unidirectional Data Transfer*, Ph.D dissertation, Dept. of Computer Science, Technical University of Berlin, 1996.

**[Schulzrinne92]** H. Schulzrinne, ed., "Issues in Designing a Transport Protocol for Audio and Video Conferences and Other Multiparticipant Real-Time Applications," *Internet Draft*, IETF, Dec. 1992.

**[Schulzrinne94]** H. Schulzrinne, S. Casner, R. Frederick & V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," *Internet Draft*, IETF, Nov. 1994.

**[Schulzrinne95a]** Henning Schulzrinne, "Operating System Issues for Multimedia," Internal Report of GMD-FOKUS, to appear in *Multimedia Systems* (R. G. Herrtwich, ed.), Springer, May 1995.

**[Schulzrinne95b]** Henning Schulzrinne, "Guide to NEVOT 3.32," GMD-FOKUS, Aug. 1995.

**[Schulzrinne96]** Henning Schulzrinne, "Real-Time Multimedia Services in the Internet," Internal Report of GMD-FOKUS, Mar. 1996.

**[Selent96]** J. Selent, *Simulation, Implementierung und Evaluierung von Echtzeitplanungsmechanismen für Multimediale Anwendungen*, Diplomarbeit, Fachgebiet für Offene Kommunikationssysteme, Fachbereich Informatik, Technische Universität Berlin, Apr. 1996.

**[Sha86]** L. Sha, J. P. Lehoczky & R. Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling," *Proc. IEEE Real-Time Systems Symposium*, IEEE, 1986.

**[Sha89]** L. Sha, et al, "Mode Change Protocols for Priority-Driven Preemptive Scheduling," *Journal of Real-Time Systems*, (1) 243-264, Kluwer Academic Publishers, 1989.

**[Sha90]** L. Sha & J. P. Goodenough, “Real-Time Scheduling Theory and Ada,” *IEEE Computer*, Apr. 1990.

**[Sha94]** L. Sha, R. Rajkumar & S. S. Sathaye, “Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems,” *IEEE Proceedings Journal*, Jan. 1994.

**[Shenker95a]** S. Shenker & C. Partridge, “Specification of Guaranteed Quality of Service,” *Internet Draft draft-ietf-intserv-guaranteed-svc-00.txt*, IETF, Mar. 1995.

**[Shenker95b]** S. Shenker & C. Partridge, “Specification of Predictive Quality of Service,” *Internet Draft draft-ietf-intserv-predictive-svc-00.txt*, IETF, Mar. 1995.

**[Shepherd92]** D. Shepherd, et al, “Protocol Support for Distributed Multimedia Applications,” *Computer Communications*, Vol. 15, No. 6, July 1992.

**[Sieckmeyer95]** C. Sieckmeyer, *Bewertung von Adaptiven Ausspielalgorithmen für Paketvermittelte Audiodaten*, Studienarbeit, Institut für Fernmeldetechnik, Technische Universität Berlin, Oct. 1995.

**[Sijben95]** P. Sijben & S. Mullender, “An Architecture for Scheduling and QoS Management in Multimedia Workstations,” *Pegasus paper 95-5*, Dept. of Computer Science, University of Twente, Dec. 1995.

**[Stankovic88]** J. A. Stankovic & K. Ramamritham, eds., *Hard Real Time Systems*, IEEE Computer Society, 1988.

**[Stankovic92]** J. A. Stankovic, “Two Extended Editorials on Real-Time Kernels and Resource Allocation,” *CMPSCI Technical Report 92-0078*, Department of Computer and Information Science, University of Massachusetts, 1992.

**[Steinmetz93]** R. Steinmetz, “Compression Techniques in Multimedia Systems: A Survey,” *Technical Report 43.9307*, IBM European Networking Center, 1993.

**[Steinmetz95]** Ralf Steinmetz & Klara Nahrstedt, *Multimedia: Computing, Communications & Applications*, Prentice-Hall PTR, 1995.

**[Sun91]** Sun, *Solaris SunOS 5.0: Multithreading and Real-Time*, a white paper of Sun Microsystems, Inc., 1991.

**[Tanenbaum87]** A. S. Tanenbaum, *Operating Systems: Design and Implementation*, Prentice-Hall, Inc., 1987.

**[Tennenhouse95]** David L. Tennenhouse, et al, “The ViewStation: a Software-

Intensive Approach to Media Processing and Distribution," *Multimedia Systems, Vol. 3,* 1995.

**[Thekkath93]** C. A. Thekkath, T. D. Nguyen, E. Moy & E. D. Lazowska, "Implementing Network Protocols at User Level," *Proc. SIGCOMM'93*, ACM, Sept. 1993.

**[TI91]** Texas Instruments, *TMS320C4x User´s Guide, 2564090-9721 Rev. A*, May 1991.

**[Tokuda89]** H. Tokuda, M. Kotera & C. W. Mercer, "An Integrated Time-Driven Scheduler for the ARTS Kernel," *Proc. 8th Annual Int'l Phoenix Conference on Computers and Communications*, IEEE, 1989.

**[Tokuda90]** H. Tokuda, T. Nakajima & P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System," *Proc. USENIX 1990 Mach Workshop*, Usenix, Oct. 1990.

**[Tokuda92]** H. Tokuda, et al, "Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network," *Proc. SIGCOMM'92*, ACM, 1992.

**[Tokuda93]** H. Tokuda & T. Kitayama, "Dynamic QOS Control based on Real-Time Threads," *Proc. 4th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video*, Nov. 1993.

**[Vetter92]** K. Vetter & R. Widyono, "Latency Measurements of a Real-Time Ultrix Kernel," *Technical Report*, ICSI & UC Berkeley, Nov. 1992.

**[Vogel95]** A. Vogel, B. Kerherve, G. von Bochmann & J. Gecsei, "Distributed Multimedia and QOS: A Survey," *IEEE Multimedia*, 1995.

**[Vogt95]** C. Vogt, "Quality-of-Service Management for Multimedia Streams with Fixed Arrival Periods and Variable Frame Sizes," *ACM Multimedia Systems*, 1995.

**[Wall92]** G. A. Wall, J. G. Hanko & J. D. Northcutt, "Bus Bandwidth Management in a High Resolution Video Workstations," *Proc. 3rd Int'l Workshop on Network and Operating System Support for Digital Audio and Video,* 1992.

**[Wittig94]** H. Wittig, L. C. Wolf & C. Vogt, "CPU Utilization of Multimedia Processes: The HeiPOET Measurement Tool," *Technical Report 43.9409*, IBM European Networking Center, 1994.

**[Wolf93]** L. C. Wolf, "Eine Laufzeitumgebung für Multimedia-Kommunikationssystems," *Informationstechnik und Technische Informatik,* 35(2), 1993.

**[Wolf96]** Lars C. Wolf, Wolfgang Burke & Carsten Vogt, "Evaluation of a

CPU Scheduling Mechanism for Multimedia Systems," *Software — Practice and Experience*, to appear, 1996.

**[Wolfinger92]** B. Wolfinger & M. Moran, "A Continuous Media Data Transfer Service and Protocols for Real-Time Communication in High Speed Networks," *Proc. 2nd Int'l Workshop on Network and Operating System Support for Digital Audio and Video*, 1991.

**[Zalewski93]** J. Zalewski, "Real-Time Systems Glossary," *Technical Report*, Dept. of Computer Science, Univ. of Texas of the Permian Basin, Dec. 1993.

**[Zeletin89]** Radu Popescu-Zeletin, "From Broadband ISDN to Multimedia Computer Networks," *Computer Networks and ISDN Systems, Vol. 18*, 1989.

**[Zhang93]** L. Zhang, et al, "RSVP: a New Resource ReSerVation Protocol," *IEEE Network*, Vol. 7, Sept. 1993.

**[Zhang94]** Lixia Zhang, "Support for Hierarchically Encoded Flows," *Working Draft of the RSVP Working Group*, IETF, Dec. 1994.

**[Zhao89]** W. Zhao & J. A. Stankovic, "Performance Analysis of FCFS and improved FCFS Scheduling Algorithms for Dynamic Real-Time Computer Systems," *Proc. Real-Time Systems Symposium*, IEEE, 1989.