C_MACSE: A Novel ACSE Protocol for Hard Real-time Multimedia Communications*

Dimitris Kanellopoulos[†] *and Sotiris Kotsiantis*[††]

[†] University of Patras, Department of Electrical & Computer Engineering, Greece
[††] University of Patras, Department of Mathematics, Greece

Summary
A connection establishment protocol for multimedia communication residing at the OSI application layer is presented. The new protocol (C_MACSE) adopts a comprehensive approach for Quality of Service (QoS), as it incorporates resource management strategies for the CPU scheduling and the virtual memory of a multimedia system. The C_MACSE protocol provides services for the negotiation, renegotiation and monitoring of the comprehensive QoS.

Key words:
Multimedia communications; QoS; ACSE; CPU scheduling; memory management.

Introduction

In a multimedia system, different media types coexist with divergent performance parameters [1], which impose strict requirements not only to the underlying network capacity and protocol efficiency, but also to the computer's ability to manage its resources effectively [2] (i.e. CPU scheduling, virtual memory and I/O devices). The end-system support for multimedia communications is required because part of the communications protocol stack is implemented in software and executed by the end-system. If the end-system cannot guarantee the execution time for instructions (referring to protocol stack), there will be no real-time communications system no matter how well networking support is provided. Besides, if the media data need to be processed (including compression and decompression) before presentation, the processing time should be predictable. Otherwise, a meaningful presentation is not achieved. The hardware should have high processing power and high data transfer throughput, because digital video and audio are very data intensive. In addition, the problem of "mismatch in bandwidth" [3] is time-critical for multimedia communications, as usual host system buses (e.g. VME bus) support lower transmission rates than high-speed networks.

In many existing communications architectures the notion of QoS is extremely narrow, because they are based on best-effort performance models and they do not support quantitative QoS. A comprehensive QoS approach considers multimedia requirements imposed both on the network and operating system. In the near past, many researchers have made specific assumptions about the end-points of multimedia communication [4]. For example, they envisaged multimedia systems equipped with specialized real-time operating systems, massive physical memory, large CPU processing power and enhancement I/O devices. However, conventional workstations running standard multiprogrammed operating systems (eg. Unix) are interfaced to ATM networks, in order to support distributed multimedia applications. In such cases, resource management strategies have to be proposed for the ATM network, but also for the CPU, the virtual memory and the I/O devices.

The main approach to developing multimedia operating systems is to modify and extend certain micro-kernels to support real-time applications, while providing a "personality" to run existing applications. A micro-kernel is an operating system that is only responsible for manipulating low-level system resources, and is independent of any specific user-level computational paradigm. Consequently, by developing hard real-time

* This paper combines and extends research work that appeared in preliminary form in Proceedings of the IEEE International Conference on Military Communications, Mc Lean, Virginia, USA, October 1996, Eds. IEEE, pp. 755-759 and in Proceedings of the 5th IEEE International Conference on Universal Personal Communications, Cambridge, Massachusetts, USA, Sept.-Oct. 1996, Eds. IEEE, pp. 487-491. Part of this work was done in RACE-Project R1086 TELEMED.

Manuscript received January 15, 2006.
Manuscript revised February 25, 2006.

multimedia communication systems, a basic consideration deals with the underlying micro-kernel. Chorus [5] is one of the most dominant micro-kernel that provides the system platform for our project. Chorus has been widely used for embedded multimedia applications, because it has real-time features: pre-emptive scheduling, page locking, system call timeouts and efficient interrupt handling. In Chorus's implementation, modern techniques such as multithreaded address spaces and Inter-Process Communication (IPC) with copy-on-write semantics have been used. The basic Chorus micro-kernel abstractions are actors, threads and ports. Protocol's developers can add new scheduling policy modules to a multimedia system via a Chorus framework called scheduling classes. For the development of communications protocols, a protocol stack consisting of a kernel-resident network device manager with a link layer interface and separate network and transport layer actors are provided. Micro-kernels such as Chorus, Mach and Amoeba have common limitations in the multimedia communications domain [6]. For example, Chorus is not directly applicable for the support of distributed multimedia applications, because it does not support QoS control and resource reservation. The performance over existing micro-kernel facilities can be improved by reducing the number of protection-domain crossings and context switches incurred. Existing Chorus abstractions can be extended in order to include QoS configurability, connection-oriented communications and real-time threads.

Our project quantifies multimedia communication requirements imposed, by the end-user, on the operating system and network and enhances the ACSE (Association Control Service Element) standard [7] taking into account system resources management policies (i.e. CPU scheduling, memory and I/O management strategies). We designed and developed an enhanced protocol (C_MACSE) that improves the ACSE standard, as it provides comprehensive QoS support and ad hoc services to multimedia application developers.

This paper is composed as follows. In Section 2, multimedia requirements imposed on the operating system are reviewed, and in Section 3, background work is discussed. Section 4 describes the comprehensive QoS architecture. The new protocol and a pilot scheduling architecture are presented in Sections 5 and 6 correspondingly. Finally, concluding remarks are summarized in Section 7.

2. Multimedia requirements in the operating system

The end-user perceives a continuous performance level for a real-time continuous media, if QoS guarantees are provided at all relevant subsystems (network, CPU, memory, I/O). In the CPU subsystem, the required QoS is expressed in terms of guaranteed CPU processing for those threads (real-time threads), which handle streams of real-time continuous media. Since continuous streams are handled by real-time threads, the kernel must deliver multimedia data directly to peripheral end-points with minimal or no interaction with the CPU. This is enforced by the real-time behavior of a continuous media stream, which implies a key requirement to the CPU scheduling of all processing threads related to this stream: "threads associated with a continuous media stream ought to be closely co-ordinated in such level such as to ensure that the temporal integrity of this media stream is not violated" [2]. In both, application and protocol processing, kernel context switches must be minimised for those threads that handle streams of real-time continuous media (real-time threads). Real-time threads must access memory regions (data, code, stack) with bounded latency. The bounded access latency levels of these threads are deduced by the enforced user-level QoS requirements, expressed for a continuous media stream.

In addition, CPU resources must be fairly shared across multiple address spaces of the processes that compose the multimedia application. The level of priority for each process varies from the QoS user's requirements. The splitting of processing demands the use of an effective CPU scheduling policy for real-time threads.

3. Background work

Three main architectures or/and models have been proposed in order the communication architecture of the Internet to provide QoS guarantees. They are IntServ [8], DiffServ [9] and MPLS [10]. Lue [11] analysed issues and technologies for supporting multimedia communications over the Internet, while Foster et al [12] described a General–purpose Architecture for Reservation and Allocation (GARA) to support secure immediate and advance co-reservation, online monitoring/control, and policy-driven management of a variety of resource types, including networks. Sampatakos et al [13] introduced a scalable inter-domain resource control architecture for Differentiated Services networks. A higher level QoS manager tool and a channel library named SALMON were proposed in [14]. In the projects HeiTS [15] and SUMO

[6], protocols and mechanisms were proposed in the domain of operating system support for handling continuous-media. The HeiTS project has investigated end-system issues in the integration of transport QoS and CPU scheduling. An elegant split scheduling scheme and IPC mechanisms for continuous media were proposed in [16]. Research work at Lancaster University [2, 6] has addressed the design of a QoS-controlled ATM-based communications system in Chorus and QoS guarantees were extended into the operating system.

In signalling protocols domain, many research trials have been made. The EXPANSE signalling protocol was introduced in [17] and a call model for the establishment of multipoint connections over ATM network was proposed in [18]. Design issues related with signalling architectures and protocols, aimed to support B-ISDN, were discussed in [19]. A novel SVC mechanism for call connection and control in ATM networks was presented in [20]. A set of generic QoS parameters that capture the varied requirements imposed by multimedia applications has been defined in the RACE Eurobridge (R 1008) project. These generic QoS parameters don’t rely on the underlying heterogeneous network infrastructures and became relevant in an enhanced ACSE element (XACSE [21]).

Our project was based on MACSE [22], a connection establishment protocol for multimedia communication residing at the OSI application layer. The objective of MACSE is the efficient establishment and control of multimedia associations among spatially dispersed users. MACSE introduced the notion of out-of-band signalling at the application layer and defined procedures for the negotiation and confirmation of QoS per association, in contrast with the structure of the well-known ACSE protocol. In the MACSE framework, the following services have been implemented: a) establishment, normal and abnormal release of multimedia associations, b) renegotiation of a particular QoS value per individual association, c) addition and deletion of users in a multimedia call, d) addition and deletion of associations in a multimedia call and e) synchronization of two or more associations in a multimedia call.

However, the main limitation of the MACSE protocol is that it does not adopt a comprehensive QoS approach, as it ignores CPU scheduling, virtual memory and I/O management issues. At the OSI application layer, connection establishment protocols have not yet adopted a comprehensive QoS approach.

4. The comprehensive QoS architecture

The C_MACSE protocol comprises an integral module in the proposed ATM-based architecture [23, 24], where service commitments are supported both in the network and in the end-system.

This architecture (depicted in Fig. 1) aspires to satisfy the requirements of hard real-time multimedia applications and presents the following advantages:

- It has mechanisms capable of negotiating, renegotiating and monitoring the provided comprehensive QoS.
- It supports synchronization services and multicast communications services.
- It does not take into consideration a revolutionary approach in resource management, and thus it operates with existing standards whenever it is feasible.

This integrated QoS architecture is composed of three vertical planes: a) the communication plane, b) the operating system plane and c) the flow management plane. The communication plane contains protocols implemented with multithread processes, while the operating system plane contains the necessary micro-kernel operating system mechanisms. Finally, the flow management plane provides services related to dynamic QoS control functions, such as flow admission control and QoS renegotiation. In particular, the proposed communication plane consists of the following modules:

1) *Application service modules*, which are application service elements (ASEs) that deliver functionalities within the OSI application layer.

2) *Session layer mechanisms*. The synchronization manager synchronizes audio and video streams, and provides session setup and tear-down as well as flexibility in QoS aspects.

3) *Transport protocol combinations*. Different transport protocol combinations such as TCP/IP, UDP/IP, or even raw sockets for IPC can be used. High-speed transport protocols such as TPX, XTP, XTPX [25] and VMTP can be used.

4) *Network protocols* support both network and end-system resource allocation (eg. RSVP [8]).

5) *The communication model of ATM* includes the AAL, the ATM layer and the physical layer. In developing ATM APIs, two different approaches can be adopted: a) LAN emulation [26] and b) classical IP over ATM [27].

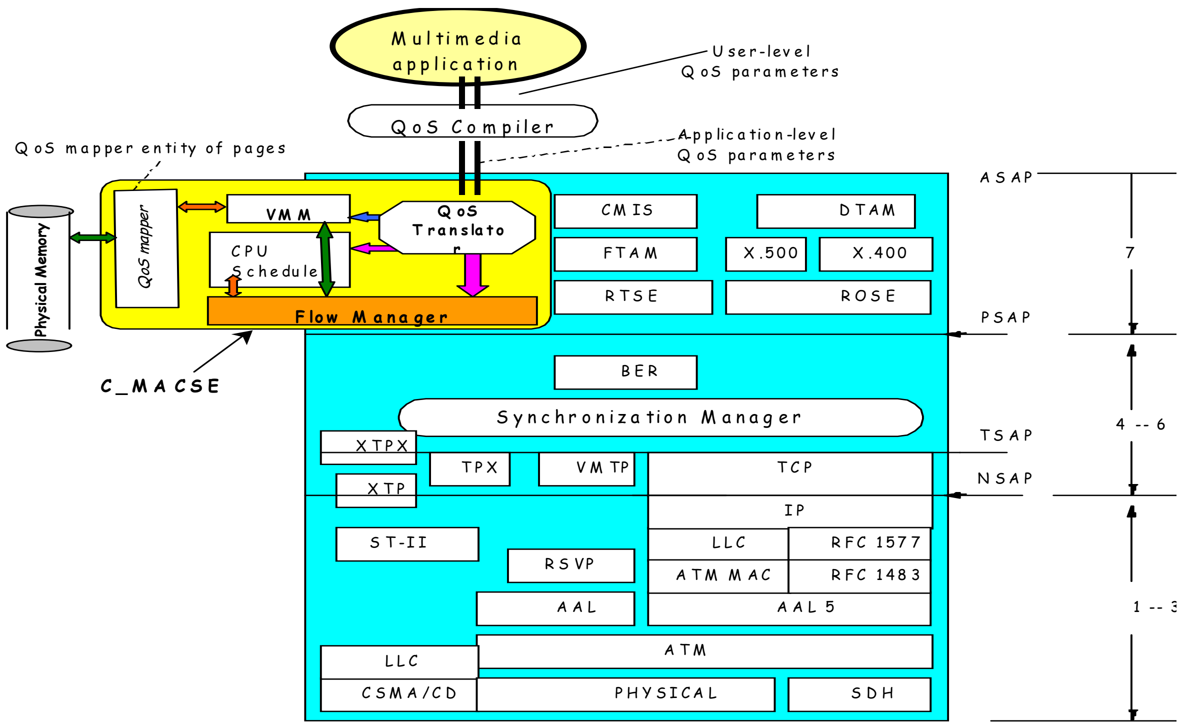

Fig. 1: The comprehensive QoS architecture [23]

5. The C_MACSE protocol

The new protocol [23, 24] is entitled: *Comprehensive approach - Multiple Association Control Service Element* (C_MACSE). It defines procedures for the support of multiple associations among multiple users and negotiation and confirmation of the comprehensive QoS per association between the peer entities. During the lifetime of an association, renegotiation of the comprehensive QoS is also supported. The ACSE standard and the MACSE can be considered as subdivided protocols of the C_MACSE protocol.

5.1 The service definition

The C_MACSE services are listed in Table 1. The C_MACSE service definition has been based on the abstract model for a service layer promoted by OSI RM.

Table 1: The C_MACSE services

Service	***Type***
C-MA-ASSOCIATE	Confirmed
C-MA-RENEGOTIATE	Confirmed
C-MA-SIGNAL-QOS	Provider-initiated
C-MA-ALTER-USER	Confirmed
C-MA-ALTER -ASSOCIATION	Confirmed
C-MA-SYNCHRONIZE	Confirmed
C-MA-RELEASE	Confirmed
C-MA-ABORT	Non-confirmed
C-MA-ABORT-USER	Non-confirmed
C-MA-ABORT-ASSOCIATION	Non-confirmed
C-MA-P-ABORT	Provider-initiated

■ `C_MA-ASSOCIATE:` Multiple associations among multiple users can be established using this service. Negotiation of a particular comprehensive QoS per individual association can also be attained. The requirement for a particular value of a QoS parameter is negotiated within the range `[acceptable, desired]` `(acceptable≤desired)` specified in the request primitive. The C_MACSE protocol machine (C_MACPM) issues a `C_MA-ASSOCIATE` `indication` primitive to every called user. The maximum value `desired` of the range may be decreased by the service provider to a new value, named `available.` The called user, in turn, may further decrease it to lower value, named `agreed` and return it with the response primitive. Calling C_MACSE user considers a multimedia call successful as soon as the first positive confirmation primitive from the C_MACSE service provider arrives. Any subsequent positive confirmation primitive results in an establishment of a new composite association. If the service provider does not support the requested comprehensive QoS (e.g. due to unavailability of network or system resources), it issues a `C_MA-SIGNAL-QOS indication` primitive back to the user indicating the type of result (negative) of the QoS negotiation.

■ `C_MA-RENEGOTIATE`: It is used to renegotiate the *comprehensive* QoS of an association during its lifetime. The C_MACSE user issues the `C_MA-RENEGOTIATE request` primitive to alter the values of the comprehensive QoS parameters initially achieved. The operation is completed by exploiting the updated properties of the association. For the coordination of the peer entities, the C_MACSE service provider issues a `C_MA-SIGNAL-QOS indication` primitive to the target user, simultaneously with a `C_MA-RENEGOTIATE confirmation` primitive to the origin. The called user considers the QoS alternation successful as soon as the former primitive arrives.

■ `C_MA-SIGNAL-QOS`. It is issued by the C_MACSE service provider to signal the result of a comprehensive QoS operation during `C_MA-ASSOCIATE` and `C_MA-RENEGOTIATE` procedures and indicate any QoS thresholds violation. When the C_MACSE service-provider detects either the inability of the underlying services (belong to network or to operating system) to support the requested QoS or a positive responded renegotiation request, it issues a `C_MA-SIGNAL-QOS indication` primitive to the corresponding user.

■ `C_MA-ALTER-USER`. It is used to provide for addition and deletion of users in a multimedia call.

■ `C_MA-ALTER-ASSOCIATION`. It is used to provide addition and deletion of associations in a multimedia call.

■ `C_MA-SYNCHRONIZE`. It is used for the activities initiation of the underlying synchronization entity. The `C_MA-SYNCHRONIZE` service `request` is issued by a C_MACSE user whenever it is desired for two or more established associations to be synchronized according to a particular relation requirement.

■ `C_MA-RELEASE` and `C_MA-ABORT` services are used by a calling application entity to cause the normal and abnormal release of a multimedia call respectively.

■ `C_MA-ABORT-USER` and `C_MA-ABORT-ASSOCIATION` services are used to provide abnormal release of an established composite association and association, respectively.

■ `C_MA-P-ABORT:` It is used by the C_MACSE service provider (i.e. network or operating system) to signal abnormal release of the multimedia call due to problems in services below the application layer.

5.2 The C_MACSE elements

The C_MACSE protocol (as depicted in Fig. 1) incorporates the following modules:

5.2.1 The QoS translator (QoS-T)

During the connection establishment or QoS renegotiation phases, user-level QoS parameters (e.g. window size, color, depth, frame rate, concerning a video stream) can be compiled into application-level QoS parameters by the *QoS compiler*. Then, QoS-T translates application-level QoS parameters into representations usable by the relevant subsystems (network, CPU and memory). To do this task, it uses tables that include relations between the two different levels of QoS parameters.

In the C_MACSE protocol, we incorporated proper QoS parameters as to provide control at the OSI application layer for network and system resource management. Concerning a media stream, these incorporated application-level QoS parameters are:

```
                 media stream_QoS =
    (commitment, delivery, buffer_rate,
 buffer_size, latency, delay priority level,
        loss priority level, jitter)
```

5.2.2 The CPU scheduler

The problem of CPU scheduling demands two steps: *a)* to classify all processing threads, generated by the multimedia application, into classes and *b)* to apply an effective scheduling policy to the same class threads. The CPU scheduler executes these tasks/steps and permits threads processing wherever required according to the notion of *urgency*. Our project is strongly based on the work presented in [6], because we use the same admission tests and resource classes for QoS-controlled connections.

Classification of threads: Threads generated from various media streams have different QoS requirements defined as *guaranteed* (G) or *best-effort* (B). The notion of *commitment* expresses a degree of certainty that the requested QoS levels will actually be granted during run-time. *Commitment* can take two values: *guaranteed* or *best-effort*. If *commitment* is *guaranteed*, physical memory and CPU resources are permanently dedicated to support the requested QoS levels of this stream. Every stream connection has two main properties: *commitment* and *delivery*. Using these properties, five resources classes are formed:

(1) G_I *:* Here, there are threads which handle streams of real-time continuous media *(*viz. *isochronous).*

(2) G_W *:* In this class belong threads, which are associated with *guaranteed* and *workahead* stream connections. The *workahead* value relaxes the restriction of a continuous media to issue an APDU at a fixed point of time. Therefore, an `A_DATA.request` or an `A_DATA.indication` service primitive may be issued at an earlier time than actually allowed.

(3) B_I*:* In this class threads are associated with *best-effort* and *isochronous* stream connections.

(4) B_W *:* In this class threads handle *best-effort* and *workahead* stream connections.

(5) B_C *:* These threads are associated with classic, non real-time, UNIX application programs.

Scheduling Policies: Threads belonging to G_I class are scheduled according to an extended *Earliest Deadline First* (EDF) algorithm [28], while threads belonging to G_W class are scheduled according to the standard pre-emptive EDF policy. In both of these thread classes, we use an admission test to ensure predictive behavior. Threads belonging to B classes are scheduled according to the pre-emptible EDF policy, but no admission test is used.

5.2.3 Virtual Memory Manager (VMM)

The VMM [23, 24] ensures that multimedia QoS-controlled connections can access memory regions with bounded latency. The time constraints are achieved by: 1) determining whether or not requests for QoS-controlled memory resources should succeed or fail and 2) preempting QoS-controlled memory resources from CPU scheduler when necessary. For example, "high urgency" threads have higher priority than "low urgency" threads. During the connection establishment or renegotiation phases, two memory-related properties are deduced from the user-level QoS parameters: a) the *number of buffers* required per stream connection and b) the *required access latency* associated with those buffers. As a thread has three access regions with the kernel, three access latency levels are considered. These levels are related to the *code*, *data* and *stack regions*. If access latency levels to these regions are bounded, QoS guarantees for the QoS-controlled threads can be achieved.

There are *swappable* and *locked* pages in memory and thus two different access latencies for these pages are observed. The latency bound of swappable pages [29] depends on: 1) the delay due to the Remote Procedure Call (RPC) communication between the VMM and the *pages mapper* (viz. a daemon process that maps the virtual memory address space to physical address space, without predictions about the pages faults of the real-time threads during their execution) and 2) on the delay associated with the external swap device. The latency bound of the locked pages depends on the *system bus* and on the *clock speed.* To bound access latency levels to swappable pages, new page replacement policies and disk layout strategies must be proposed. Page replacements algorithms oriented to the "*working set model*" [30] are the most promising. Resultantly, adjusting the above system parameters, access latency levels can be bounded. In the C_MACSE framework, we designed and developed a QoS *pages mapper* that is based on the *reference* and *distance string* models [31]. QoS guarantees (relating to page faults of real-time threads) are achieved partially.

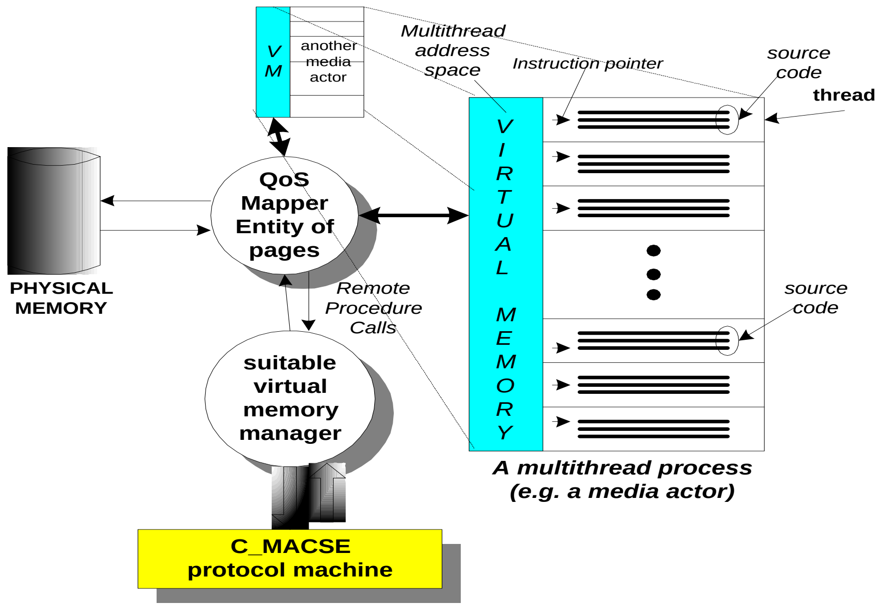
V M
another media actor
Multithread address space
Instruction pointer
source code
thread
V I R T U A L M E M O R Y
QoS Mapper Entity of pages
PHYSICAL MEMORY
Remote Procedure Calls
source code
suitable virtual memory manager
A multithread process (e.g. a media actor)
C_MACSE protocol machine

Fig. 2: The memory management architecture [23]

5.2.4 The flow manager (FM)

FM integrates the resource management in both communication end-systems and the network. Precisely, FM is supervising the three resource subsystems and executes three admission tests (i.e. *bandwidth test*, *delay bound test*, and *buffer availability test*), in order the C_MACSE protocol to determine whether a new session can be created, given its specific resource requirements and the availability of the subsystems resources. During connection time, FM arranges the memory and allocates proper CPU and network resources. It uses a prior resource reservation mechanism to obtain guaranteed real-time performance. In addition, FM is responsible for dynamic QoS management and adapts the degradations made in one subsystem resource by compensating in terms of the others.

6. A pilot CPU scheduling architecture

In order to meet multimedia application requirements, we developed a pilot split-level CPU scheduling architecture [23] in the SunOS R.4.1 operating system. This architecture resides in the user-level space, as light-weight processes (LWP) library lacks kernel support. It exploits the concept of light-weight threads and classifies all processing threads generated by the current multimedia application. It applies the scheduling policies (described above) to the same class threads and permits threads CPU processing, where required, according to the notion of *urgency*. In this architecture, there are *media*, *network* and *memory* actors (e.g. QoS mapper). The network actor contains threads implementing the OSI upper layer protocols of the communication architecture [24]. Inside the network actor, threads are organized according to the pipe-line model. This architecture is split in two main levels: a) the level of the application level scheduler (ALS) and b) the level of User Level Schedulers (ULSs). The ALS communicates with the ULSs using shared memory and software interrupts. The common shared memory is divided in parts that contain the current value of *urgency* for each ULS. The *urgency* of a ULS expresses the priority of the most urgent thread running above this ULS. Therefore, this architecture always executes the most urgent thread, while the value of *urgency* is deduced indirectly by the user-level QoS parameters using the QoS_T.

The main factor that provoked potential violations of the scheduling invariants in this architecture was the execution of blocking system calls by real-time threads. We solved this problem by using the non-blocking I/O library (`libnbio.a`) and rewriting parts of the system calls library. Particularly, we placed code (*jacket*) around the “suspicious” system calls to do the checking.

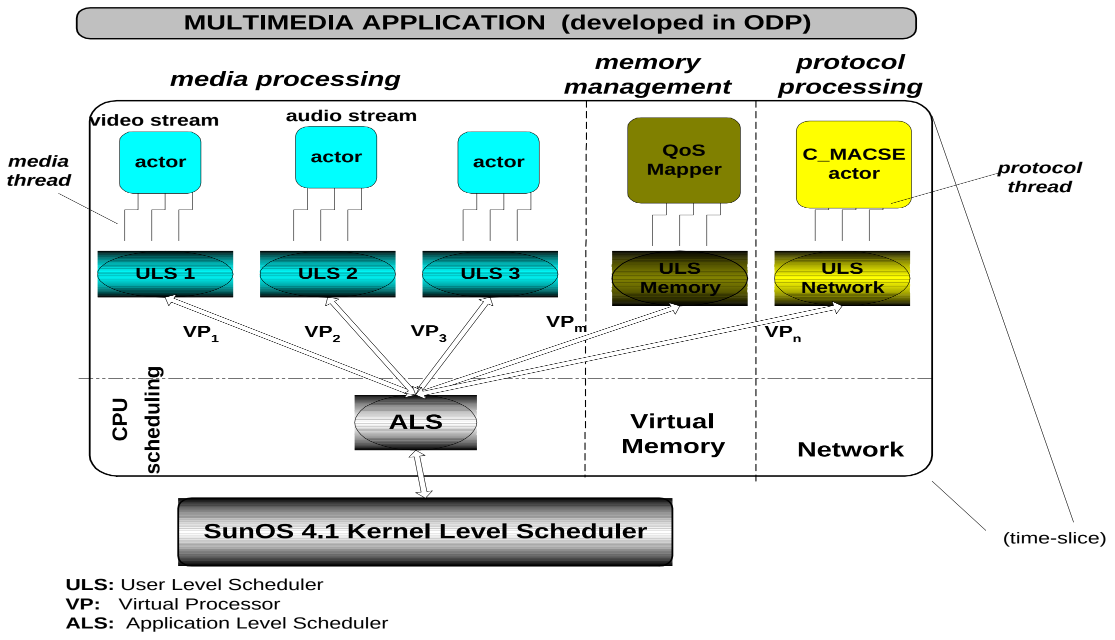

Figure 3: The pilot scheduling architecture in SunOS R.4.1 [23]

7. Conclusion

The field of connection establishment protocols has not yet adopted a comprehensive QoS approach in development and standardization. An important advantage of the proposed C_MACSE protocol is that it robustly and effectively reflects not only the network capabilities, but also the operating system resource management strategies by incorporating specific mechanisms for threads scheduling and virtual memory management. An integrated communication architecture with comprehensive QoS support for ATM-based multimedia communications systems, as well a pilot CPU scheduling architecture have been proposed.

References

[1] H. Chu, K. Nahrstedt, CPU service classes for multimedia applications, *Proc.IEEE Conf. on Multimedia Computing and Systems*, 1999.

[2] G. Coulson, G. Blair, Micro-kernel support for continuous media in distributed systems, *Computer Networks & ISDN Systems*, *26*, 1994, 1323-1341.

[3] Special Issue: *End-System Support for High-Speed Networks/Breaking the Network I/O Bottleneck, IEEE Network*, *7*(4), 1993.

[4] M. Zafirovic-Vukotic, I. Niemegeers, Multimedia Communication Systems: Upper Layers in the OSI Reference Model, *IEEE J. Select. Areas Commun.*, *10*(9), 1992, 1397-1402.

[5] M. Rozier *et al.*, Overview of the CHORUS Distributed Operating Systems, *Computing Systems, Journal of the UNIX Association*, *1*(4), 1991.

[6] G. Coulson *et al.*, The Design of a QoS-Controlled ATM-Based Communications System in Chorus, *IEEE J. Select. Areas in Communications*, *13*(4), 1995, 686-699.

[7] CCITT Recommendation X.217: 'Association Control Service Definition for Open Systems Interconnection for CCITT applications' (Vol. VIII, Fascicle VIII.4, 1988).

[8] R. Braden *et al.*, Resource ReSerVation Protocol (RSVP)-Version 1 Functional specification, Internet RFC2205, September 1997.

[9] Y. Nernet *et al.*, A framework for differentiated services, IETF Internet Draft: draft-diffserv-framework-02.txt, February 1999.

[10] G. Armitage, MPLS: the magic behind the myths, *IEEE Commun. Magazine, 38*(1), 2000, 124-131.

[11] G. Lu, Issues and technologies for supporting multimedia communications over the Internet, *Computer Communications, 23*, 2000, 1323-1335.

[12] I. Foster *et al.*, End-end quality of service for high-end applications, *Computer Communications*, 2004 (to be published).

[13] P. Sampatakos *et al*, BGRP: Quite Grafting Mechanisms for Providing a Scalable End-to-End QoS solution, *Computer Communications, 27*, 2004, 423-433.

[14] A. Schill, T. Hutschenreuther, Architectural support for QoS management and abstraction: SALMON (Support

Architecture for transmission of Live Media streams On networks), *Computer Communications, 20*, 1997, 411-419.

[15] D.B. Hehmann *et al.*, Implementing HeiTS: Architecture and implementation strategy of the Heidelberg high-speed transport system, *Proc. 2nd Int. Workshop Network, Operating Sys. Support Digital Audio, Video,* 1991, (Heidelberg Germany).

[16] R. Gonindan, D.P. Anderson, Scheduling and IPC mechanisms for continuous media, *in Proc. Thirteenth ACM Symp. Operating Syst. principles* (Pacific Grove, CA), SIGOPS, *25*, 1991, 68-80.

[17] S. Minzer, A Signaling Protocol for Complex Multimedia Services, *IEEE J. Select. Areas Commun.*, 9(9), 1991, 1383-1394.

[18] R. Bubenik *et al.,* Multipoint connection management in high-speed networks, *Proc. IEEE INFOCOM,* 1991 59-68.

[19] T. La Porta *et al.,* B-ISDN: A technological discontinuity, *IEEE Commun. Mag., 32*(10), 1994, 84-97.

[20] R. Henry, P. Darby, A novel SVC mechanism for call connection and control in ATM networks, *Computer Communications, 20,* 1997, 1123-1128.

[21] D. Mc Glinchey, XACSE: Connection Set-up for Broadband Services using Generic QoS Parameters, *Proc. of the 2nd Brodband Islands Conf.*, 1993, 207-210.

[22] G. Orphanos, G. Papadopoulos, S. Koubias, MACSE: A Generic Connection Establishment Protocol for Multimedia Communication Residing at the OSI Application Layer, *European Transactions on Telecommunications*, *9*(3), 1998, 295-311.

[23] D. Kanellopoulos, S. Koubias, G. Papadopoulos, A Novel User-to-User Protocol for the Establishment of Multimedia Associations adopting a Comprehensive QoS Approach", *Proc. IEEE Conf. on Military Communications (MILCOM),* 1996, 755-759.

[24] D. Kanellopoulos, G. Papadopoulos, S. Koubias, A Novel ACSE Protocol with Comprehensive QoS Support for Multimedia Communications in Chorus", *Proc. 5th IEEE Conf. on Universal Personal Communications*, 1996, 487-491.

[25] Specification of the Broadband Transport Protocol XTPX, CIO/RACE 2060, 1994.

[26] J. Heinanen, Multiprotocol Encapsulation over ATM Adaptation Layer 5, Networking Working Group, *RFC 1483, Telecom Finland*, July 1993.

[27] M. Perez *et al.,* ATM Signalling Support for IP over ATM, *RFC 1755,* February 1995.

[28] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, *J. Assoc. Computing Mach., 20*(1), 1973, 46-61.

[29] A. S. Tanenbaum, Modern Operating Systems, Vol. I, Prentice-Hall (Ed.), 1992.

[30] K. Park *et al*., The working set algorithm has competitive ratio less than two, Information Processing Letters, 63, 1997, 183-188.

[31] B. Prieve, R, Fabry, VMIN: an optimal variable space page replacement algorithm, *Communications of the ACM, 19*(5), 1976, 295-297.

Dimitris Kanellopoulos received a diploma in electrical engineering and a Ph.D. degree in electrical and computer engineering from the University of Patras, Greece. Since 1990, he was a research assistant in the Department of Electrical and Computer Engineering at the University of Patras and involved in several EU R&D projects (e.g. RACE I and II, ESPRIT). His research interests are in the field of hard real-time multimedia communication protocols, ATM networking, new OSI services and web engineering. He has more than 30 publications to his credit in international journals and conferences in these areas. Dr. Kanellopoulos is a member of the Technical Chamber of Greece.

Sotiris Kotsintis received a diploma in mathematics, a Master and a Ph.D. degree in computer science from the University of Patras, Greece. His research interests are in the field of data and multimedia mining. He has more than 40 publications to his credit in international journals and conferences.

REMARK: THIS APPENDIX WILL BE USED ONLY FOR THE REVIEWING PROCESS (NOT FOR PUBLISHING)

APPENDIX A: C_MA-ASSOCIATE service primitives and a portion of the C_MACSE protocol specification in ASN.1 notation

SERVICE PRIMITIVE	PARAMETER
C_MA-ASSOCIATE request	Multimedia-call-identifier Calling-AE *sequence* {Called-AE Composite-association-identifier User information *sequence* {Association-identifier Application-context-name **commitment** or service type **delivery** **QoS_{NET}** throughput *acceptable* *desired* transit-delay *acceptable* *desired* transit-delay-jitter *acceptable* *desired* residual-error-rate *acceptable* *desired* **QoS_{CPU}** **period** **quantum** *acceptable* *desired* **QoS_{MEM}** **buffers** *acceptable* *desired* **buffer_size** *acceptable* *desired* } *association-list* } *user-list* *sequence* {Synchronization-identifier Synchronization-type *sequence* {Application-context-name } } *synchonization-relations (U)*

C_MA-ASSOCIATE indication	Multimedia-call-identifier Calling-AE Called-AE Composite-association-identifier User information *sequence* {Association-identifier Application-context-name **commitment** or service type **delivery** **QoS$_{NET}$** throughput *acceptable* *available* transit-delay *acceptable* *available* transit-delay-jitter *acceptable* *available* residual-error-rate *acceptable* *available* **QoS$_{CPU}$** **period** **quantum** *acceptable* *available* **QoS$_{MEM}$** **buffers** *acceptable* *available* **buffer_size** *acceptable* *available* } *association-list* *sequence* {Synchronization-identifier Synchronization-type *sequence* {Application-context-name } } *synchronization-relations (U)*

SERVICE PRIMITIVE	PARAMETER
C_MA-ASSOCIATE response	Multimedia-call-identifier Called-AE Composite-association-identifier User information *sequence* {Association-identifier Application-context-name **QoS$_{NET}$** throughput *agreed* transit-delay *agreed* residual-error-rate *agreed* **QoS$_{CPU}$** **period** **quantum** *agreed* **QoS$_{MEM}$** **buffers** *agreed* **buffer_size** *agreed* Association-result Association-diagnostic } *association-list* Synchronization-result Result Diagnostic

SERVICE PRIMITIVE	PARAMETER
C_MA-ASSOCIATE confirmation	Multimedia-call-identifier Called-AE Composite-association-identifier *sequence* {Association-identifier Application-context-name **QoS$_{NET}$** throughput *agreed* transit-delay *agreed* residual-error-rate *agreed* **QoS$_{CPU}$** **period** **quantum** *agreed* **QoS$_{MEM}$** **buffers** *agreed* **buffer_size** *agreed* Association-result Association-diagnostic } *association-list* Synchronization-result Result Result-source Diagnostic

For notational reasons: *sequence* denotes one or more elements
U denotes that the presence of the parameter is user-optional

```
C_MACSE-1-0 DEFINITIONS ::=

-- C_MACSE-1-0 refers to version 1.0 for the MACSE protocol

BEGIN

-- The following elements define the APDUs used in the C_MACSE protocol

C_MACSE-apdu ::= CHOICE {
        masrq   [APPLICATION 0]   C_MASRQ-apdu,
        masre   [APPLICATION 1]   C_MASRE-apdu,
        mrnrq   [APPLICATION 2]   C_MRNRQ-apdu,
        mrnre   [APPLICATION 3]   C_MRNRE-apdu,
        maurq   [APPLICATION 4]   C_MAURQ-apdu,
        maure   [APPLICATION 5]   C_MAURE-apdu,
        macrq   [APPLICATION 6]   C_MACRQ-apdu,
        macre   [APPLICATION 7]   C_MACRE-apdu,
        msnrq   [APPLICATION 8]   C_MSNRQ-apdu,
        msnre   [APPLICATION 9]   C_MSNRE-apdu,
        mrsrq   [APPLICATION 10]  C_MRSRQ-apdu,
        mrsre   [APPLICATION 11]  C_MRSRE-apdu,
        mabrt   [APPLICATION 12]  C_MABRT-apdu
}

-- C_MA-ASSOCIATE service APDUs

C_MASRQ-apdu ::= SEQUENCE {
        protocol-version             [0]     Version-type,
        calling-AE                   [1]     AE-type,
        user-list                    [2]     User-type,
        synchronization-relations    [3]     Relations-type
}

C_MASRE-apdu ::= SEQUENCE {
        protocol-version             [0]     Version-type,
        called-AE                    [1]     AE-type,
        association-list             [2]     Association-response-type,
        synchronization-result       [3]     Sync-result-type,
        result                       [4]     Result-type,
        diagnostic                   [5]     Diagnostic-type
}

-- C_MA-RENEGOTIATE service APDUs

C_MRNRQ-apdu ::= SEQUENCE {
        calling-AE                   [0]     AE-type,
        called-AE                    [1]     AE-type,
        qos                          [2]     Qos-type
}

C_MRNRE-apdu ::= SEQUENCE {
        called-AE                    [0]     AE-type,
        qos                          [1]     Qos-response-type
}

-- C_MA-ALTER-USER service APDUs

C_MAURQ-apdu ::= SEQUENCE {
        calling-AE                   [0]     AE-type,
        user-addition-list           [1]     User-type,
        user-deletion-list           [2]     User-deletion-type,
        user-deletion-reason         [3]     Deletion-reason-type
}
```

```
C_MAURE-apdu ::= SEQUENCE {
            called-AE                       [0]     AE-type,
            association-list                [1]     Association-response-type,
            result                          [2]     Result-type,
            diagnostic                      [3]     Diagnostic-type,
            user-deletion-result            [4]     User-deletion-result-type
}

-- C_MA-ALTER-ASSOCIATION service APDUs

C_MACRQ-apdu ::= SEQUENCE {
            calling-AE                      [0]     AE-type,
            called-AE                       [1]     AE-type,
            association-addition-list       [2]     Association-type,
            synchronization-relations       [3]     Relations-type,
            association-deletion-list       [4]     Association-deletion-type,
            association-deletion-reason     [5]     Reason-type
}

C_MACRE-apdu ::= SEQUENCE {
            called-AE                       [0]     AE-type,
            association-addition-result-list [1]    Addition-result-type,
            synchronization-result          [2]     Sync-result-type,
            result                          [3]     Result-type,
            diagnostic                      [4]     Diagnostic-type,
            association-deletion-result     [5]     Deletion-result-type
}

-- C_MA-SYNCHRONIZE service APDUs

C_MSNRQ-apdu ::= SEQUENCE {
            calling-AE                      [0]     AE-type,
            called-AE                       [1]     AE-type,
            synchronization-relations       [2]     Sync-relations-type
}

C_MSNRE-apdu ::= SEQUENCE {
            called-AE                       [0]     AE-type,
            synchronization-result          [1]     Sync-result-type
}

-- C_MA-RELEASE service APDUs

C_MRSRQ-apdu ::= SEQUENCE {
            reason                          [0]     Reason-type,
            user-information                [1]     User-info-type
}


C_MRSRE-apdu ::= SEQUENCE {
            reason                          [0]     Reason-type,
            user-information                [1]     User-info-type,
            result                          [2]     Result-type
}

-- C_MA-ABORT service APDUs

C_MABRT-apdu ::= SEQUENCE {
            user-information                [0]     User-info-type,
            abort-source                    [1]     Source-type
}

-- 1st level of supplementary definitions

Version-type ::= INTEGER
```

```
AE-type ::=  OCTET STRING

User-type ::= SEQUENCE {
            called-AE                        [0]        AE-type,
            association-list                 [1]        Association-type
}

Association-response-type ::= SEQUENCE {
            application-context-name         [0]        Context-type,
            qos                              [1]        Qos-response-type
}

Relations-type ::= SEQUENCE {
            synchronization-type             [0]        Sync-type,
            application-context-name         [1]        Context-names-type
}

Qos-type ::= SEQUENCE {
            throughput                       [0]        Throughput-type,
            transit-delay                    [1]        Delay-type,
            residual-error-rate              [2]        Rer-type
}

Qos-response-type ::= SEQUENCE {
            throughput                       [0]        Throughput-response-type,
            transit-delay                    [1]        Delay-response-type,
            transit-delay-jitter             [3]        Jitter-type
            residual-error-rate              [2]        Rer-type
}

Sync-relations-type ::= SEQUENCE {
            synchronization-type             [0]        Sync-type,
            association-ids                  [1]        Ids-type
}

Association-deletion-type ::= SEQUENCE {
            assoc-id                         [0]  INTEGER
}

User-info-type ::= ANY

-- 2nd level of supplementary definitions

Association-type ::= SEQUENCE {
            application-context-name         [0]        Context-type,
            user-information                 [1]        User-info-type,
            qos                              [2]        Qos-type
}

Context-type ::= OBJECT IDENTIFIER

Sync-type ::= CHOICE {
            temporal                         [0]  INTEGER,
            spacial                          [1]  INTEGER,
            logical                          [2]  INTEGER
}

Context-names-type ::= SEQUENCE {
            application-context-name         [0]  Context-type
}

Ids-type ::= INTEGER

-- 3rd level of supplementary definitions

Throughput-type ::= CHOICE {
```

```
            acceptable                      [0]         INTEGER,
            desired                         [1]         INTEGER
}

Delay-type ::= CHOICE {
            acceptable                      [0]         INTEGER,
            desired                         [1]         INTEGER
}

Rer-type ::= INTEGER

Jitter-type ::= INTEGER

Throughput-response-type ::= INTEGER {
            agreed(0)
}

Delay-response-type ::= INTEGER {
            agreed(0)
}

END
```