μChoices: An Object-Oriented Multimedia Operating System

Roy H. Campbell See-Mong Tan
Department of Computer Science
University of Illinois at Urbana-Champaign
Digital Computer Laboratory
1304 W. Springfield
Urbana, IL 61801
{*roy,stan*}*@cs.uiuc.edu*

Abstract

This paper describes the design of the μChoices *object-oriented multimedia operating system.* μChoices *provides an architecture for interconnecting different OS subsystems, with these subsystems realized as separate modules. The modules will be implemented as independent object-oriented frameworks. Frameworks interact through exported abstract interfaces. The subclassing of components within frameworks enables application and media-specific customization.* μChoices *also provides a unified scheme for memory handling and passing across, as well as between, all OS subsystems. This allows buffer transfers and manipulation within and between operating system modules without copying, while allowing subsystems to specialize their views of memory buffers for efficient handling of problem-specific behavior. Interpreted agents may be embedded in the kernel that can control system level processing of multimedia streams without interference, eliminating excessive system call overhead. Operating system support for authentication, encryption, and delegation is transparently provided via an extensible framework that customizes interfaces to operating system resources. A new networking subsystem based on an Asynchronous Transfer Mode network environment will allow Quality of Service guarantees within the network protocol stack. These features are combined in* μChoices *to give an environment that will support high bandwidth multimedia streams.*

1 Introduction

μChoices is an object-oriented operating system geared toward supporting high bandwidth multimedia streams. The operating system is targeted toward uniprocessor as well as small to medium scale multiprocessor (2 to 32 processor) machines. *μChoices* draws many of its design ideas from lessons learned in our work on the original *Choices*[5, 4] operating system, while incorporating several new innovative ideas. This paper presents our design of *μChoices*.

A multimedia capable operating system must support a wide range of traffic types. Video and audio data streams are likely to be pervasive in the network and compute fabric of the future. We designed *μChoices* to support ubiquitous video and audio streams. Previous work on multimedia support in operating systems[27, 3, 14, 17] have concentrated primarily on scheduling algorithms for continuous media. Our work will complement the existing corpus by providing frameworks for organizing the various mechanisms and optimizations. In addition, we examine how the architecture of the operating system can be tuned to support high bandwidth multimedia (especially video) streams. Supporting one multimedia stream is not difficult, the hard part is scaling the operating system to support and orchestrate many different streams at once. In order to increase scalability, we take efforts to reduce second order effects such as those due to extraneous memory copy operations.

A multimedia operating system must handle a variety of continuous and variable bit rate streams of data, each with different Quality of Service requirements. It is unlikely that a single mechanism or policy for handling every variation in class of traffic type can be built into the operating system, so we have purposely designed the system such that components and frameworks may be extended and specialized to accommodate problem-specific behavior. In previous work, we showed that application-specific customizability can significantly improve the performance of regular scientific applications[19]. We extend our work to traffic-specific and media-specific customization.

Continuous data streams have a strong impact on memory resource management. Disk blocks or network packets containing multimedia data streams flow through operating system modules, for example, a video server may repeatedly read a disk block containing a video frame and send it out over the network, while a video client may read video disk blocks and send it to the display subsystem. *μChoices* is designed to provide low latency data paths between all OS modules by unifying memory buffer handling across the entire operating system, thus eliminating the common penalty incurred with copying due to changing memory buffer abstractions.

2 Modules and Frameworks

μChoices's design is based on a framework for interconnecting different OS subsystems, with these subsystems realized as separate modules. Modules will

be implemented as independent object-oriented frameworks that interact through well defined interfaces. Within a framework, the subclassing of components will allow us to customize its various parts to support both application and media-specific specialization. *μChoices* may be viewed as a framework for embedding the sub-frameworks for each OS subsystem.

Our previous work on *Choices* has shown that it is feasible to build an operating system through object-oriented techniques of encapsulating all system resources as objects. Abstract classes are used to capture the properties of system resources, while subclasses implement the actual details for particular machines or processors. This achieves portability across different platforms. A collection of abstract classes may cooperate to form a framework for a particular OS subsystem[6].

We envision *μChoices* to have *independent* subframeworks for each OS subsystem. That is, each OS subsystem is a module realized as a set of abstract classes. Sound software engineering principles may be used to decompose the operating system into interacting modules. Modules may interact only through well-defined interfaces. While code reuse through inheritance is encouraged internally within a module, we do not allow inheritance across sub-frameworks. This design effectively decouples each module from implementation details specific to other modules. We consider inheritance across module boundaries ill-advised, leading to complicated dependencies and interrelationships between classes in different modules.

3 Customization for Multimedia Streams

Replacing traditional operating system communication and scheduling mechanisms with customized implementations can improve the performance of applications [18]. We used object-oriented frameworks to organize the large range of optimizations possible in the *Choices* operating system.

Multimedia data streams have a wide range of characteristics — high bandwidth constant and variable bit rate video, low bandwidth audio or compact disc quality audio. Different traffic types require specialized operating system support in order to maintain quality of service guarantees[17]. For example, different scheduling disciplines and resource allocation strategies are required for different traffic classes.

We extend our previous ideas on application-specific customization through the use of object-oriented frameworks. Components in the framework may be subclassed in *μChoices* to customize behavior for different classes of multimedia data streams. For example, different subclasses of the system schedulers and resource allocators may be implemented to support isochronous constant bit rate, variable bit rate and available bit rate data. Buffer handling for the networking subsystem may be specialized toward large numbers of small cells for Asynchronous Transfer Mode (ATM) connections, or small numbers of large messages in the case of IP over an Ethernet.

4 Agents

Operating system design has moved from monolithic, single kernels such as Unix[24] where all OS services are implemented in the kernel, to micro-kernel designs such as Mach[23], where the majority of OS services are user-space applications. The communication overhead incurred when invoking server operations in micro-kernel designs is often significantly high[21]. For performance reasons, newer kernels allow the migration of some services back into the kernel, alleviating the need for multiple cross domain interprocess communication calls. What services exist in the kernel and what are outside it is a configuration decision (eg. Spring[15]). With newer RISC machine architectures built around deep instruction pipelines, a user trap into the kernel to access operating system services is becoming more and more expensive to service.

In designing the Jetstream[10] LAN, Edwards et al. amortize system call overhead over several operations by *batching* operations into a script. Recent systems such as SPIN[2] allow user applications to insert properly verified application code sequences into the kernel to customize the operating system for application-specific operating system services. Our approach is to embed interpreted *agents* into the kernel. These agents can be checked for security purposes in the same way SPIN verifies its application code sequences. Agents may be built to aggregate kernel calls for application programs that frequently call the system. For example, the Unix "ls" or "find" commands spend most of their time accessing the file system. Agents may be used to remove control traffic between the user and kernel. Trusted agents may be used to provide kernel level processing of multimedia streams between different transport, storage, and display devices. For example, it is usually unnecessary to pass a video frame coming over the network to application space before displaying it. Instead, the agent may control system processing of video frames arriving from network subsystem to display subsystem without interference or crossing protection domains.

We envision agents to be implemented in a simple, flexible scripting language similar to Tcl[22]. We are experimenting with different ways for efficient execution of interpreted agent scripts.

5 Unified Buffer Management

μChoices is composed of independent modules interacting through well specified module interfaces. While the passing of integer and pointer arguments between modules may be accomplished with call-by-value, the passing of memory buffers is more problematic. Ideally, memory buffers should not be copied when passed as arguments between different OS subsystems, or to user applications. The memory copying overhead on networking paths from user to networking device and vice versa is well known to impact negatively on network throughput and latency[7]. However, different modules of traditional operating systems often have different abstractions for memory buffers. For example, the disk subsystem may operate on buffers of disk blocks, while the networking sub-

system may operate on network messages or packets. Passing a memory buffer from the disk to the network subsystem (for example, when a video server delivers a video frame stored on disk over the network) necessitates a copy from a disk block buffer to a network packet buffer as the cost of changing buffer abstractions. Memory buffer copying increases the CPU-memory bus utilization. It may become a severe bottleneck for a system intended to handle continuous streams of multimedia data. This is an unnecessary penalty that can be eliminated if all subsystems use a single, unified scheme for handling memory buffers. Memory buffers may then be passed from one module to another without copying.

The design of *μChoices* includes a scheme for memory buffer manipulation and passing across and within all OS subsystems. Our idea unifies memory use across the entire operating system by reifying memory buffers (regions of memory). Memory buffers are represented as *MemoryObjects*, which may be thought of as generalizations of Unix mbufs[1]. Thus a physical page frame may be represented as a *MemoryObject*. Either swapping the page to disk or sending it out over the network requires no copying or format conversions since the virtual memory, disk and network subsystems all operate on the same abstraction.

While *MemoryObjects* may serve as the common currency for memory buffers within the operating system, it is doubtful that a single abstraction will suffice to support the requirements of every subsystem. Thus we allow different *views* to be taken of the same underlying *MemoryObject*. *MemoryObjectViews* aggregate *MemoryObjects* and impose a subclass specific structure on the *MemoryObjects*. Subsystems may specialize their views of memory buffers for efficient handling of problem-specific behavior. For example, the networking subsystem would have a *MemoryObjectView* subclass allowing efficient addition of network headers and trailers. The *MemoryObjectView* class must support the *BECOMES* relation, to allow one view to become another on transfer from one module to another. All representations of a *MemoryObject* will allow iteration from one memory buffer region to the next, thus conversion from one view to another is straightforward. This concept may also be supported through a meta-object protocol. We are currently considering the merits of each approach.

This idea negates the penalty imposed by changing memory buffer abstractions across modules within the operating system. It is possible for particular subclasses of *MemoryObject* to allow the fast cross-domain page remapping strategy used in Fbufs[9]. Memory buffer allocation needs to be aligned within a protection domain, otherwise page remapping may cause fragments of a memory buffer for another domain be mapped into the virtual memory space of another domain. Our unified scheme can provide a low latency path for passing memory buffers between operating system modules, as well as user space applications.

6 Security and Authentication

μChoices will provide a secure environment through the use of authentication, encryption, and delegation. The access control model of security [20] extended with compound principals[12] will be used throughout the system. As with other subsystems of *μChoices*, security is implemented through a customizable framework [26], which provides for authentication and encryption without requiring restructuring of applications. Implementation of the security framework of *μChoices* will take advantage of the *MemoryObject* class, using subclassed *MemoryObjectViews* to provide encryption, authentication, and any other security features necessary to the user.

We are also investigating methods of increasing performance of security operations to allow greater integration and use with multimedia data. The use of delegation and authentication methods allow trading of current CPU time for future performance improvement of secure sessions, such as found in [13].

7 Network Subsystem

The networking subsystem forms an integral part of a multimedia operating system. However, traditional networking subsystems are strongly IP-centric. The data structures and scheduling policies within traditional OS kernels are tuned toward supporting IP. Message subsystems are optimized for small numbers of large messages, not large numbers of small cells, as is the case for ATM networks. All data streams are also equivalent — there is no comparable concept of "bandwidth allocation." With desktop workstations in the near future likely to be small to medium scale multiprocessors, the networking subsystem must be designed to take advantage of multiprocessing. Current multiplexed protocol stacks are unsuited for efficient multiprocessing. Message multiplexing at each protocol layer requires locking of shared protocol data structures, leading to undesirable blocking and potential problems with priority inversion[25].

μChoices is designed with an ATM network environment in mind. ATM provides virtual connections between network nodes. We treat ATM virtual connections as representing end-to-end application communication. Early demultiplexing[8, 11] based on ATM virtual path and circuit identifiers can identify data streams once they arrive at the network interface. Message handling for different streams within the network subsystem can then inherit the Quality of Service parameters from the application.

Having explicit session objects in the networking subsystem (eg. Peterson and Hutchinson's work with the *x*-kernel[16]) together with early demultiplexing can lead to a network subsystem architecture with all data streams decoupled from one another. This is advantageous for multiprocessor implementation.

8 Conclusion

New operating systems are required to handle the pervasive video and audio streams of the near future. We have addressed these issues by designing the *μChoices* operating system around the idea of independent, object-oriented frameworks that encourage application-specific customizability of operating system services and policies. Components in the frame-

works may be specialized to accommodate multimedia traffic streams with different characteristics and requirements.

μChoices includes interpreted agents that can control system level processing of multimedia streams within the kernel. Doing so eliminates the overhead of cross-domain user to kernel system call invocations for operating system services when a scripted agent can do the job. It also provides more flexibility than simple batching[10].

The unified memory buffer management scheme for *μChoices* will allow the manipulation and passing of memory buffers between subsystems without copying. This allows low latency data paths between system modules.

We intend to experiment with schemes in which applications may request and receive Quality of Service assurances from the operating system. Policies and mechanisms are required to ensure smooth degradation of multimedia applications under load. Early demultiplexing in a ATM network environment will enable Quality of Service guarantees to be extended within the network protocol stack to end-to-end network data streams.

Combining these three features in *μChoices* allow it to support video and audio data streams in the ubiquitous multimedia environment of the future.

References

[1] Maurice J. Bach. *The Design of the UNIX Operating System*. Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[2] B. N. Bershad, C. Chambers, S. Eggers, C. Maeda, D. McNamee, P. Pardyak, and S. Savage amd E. G. Sirer. SPIN: An extensible microkernel for application-specific operating system services. Technical Report 94-03-03, Department of Computer Science, University of Washington, February 1994.

[3] A. Campbell, G. Coulson, F. Garcia, D. Hutchinson, and H. Leopold. Integrated Quality of Service for Multimedia Communications. In *COMM '92*, pages 99–110, 1992.

[4] Roy Campbell, Nayeem Islam, Peter Madany, and David Raila. Designing and Implementing Choices:an Object-Oriented System in C++. *Communications of the ACM*, September 1993.

[5] Roy H. Campbell and Nayeem Islam. " Choices: A Parallel Object-Oriented Operating System". In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*. MIT Press, 1993.

[6] Roy H. Campbell, Nayeem Islam, Ralph Johnson, Panos Kougiouris, and Peter Madany. *Choices*, Frameworks and Refinement. In Luis-Felipe Cabrera and Vincent Russo, and Marc Shapiro, editor, *Object-Orientation in Operating Systems*, pages 9–15, Palo Alto, CA, October 1991. IEEE Computer Society Press.

[7] P. Druschel, M. B. Abbot, M. A. Pagels, and L. L. Peterson. Network Subsystem Design. *IEEE Network Magazine*, July 1993.

[8] P. Druschel and L. Peterson. Experiences with a High-Speed Network Adaptor: A Software Perspective. In *SIGCOMM '94*, August 1994.

[9] P. Druschel and L. L. Peterson. Fbufs: A high-bandwidth cross domain transfer facility. In *Fourteenth ACM Symposium on Operating Systems Principles*, pages 189–202, Dec 1993.

[10] A. Edwards, G. Watson, J. Lumley, D. Banks, C. Calamvokis, and C. Dalton. User-space protocols deliver high performance to applications on a low-cost Gb/s LAN. *SIGCOMM '94*, August 1994.

[11] D. C. Feldmeier. Multiplexing issues in communication system design. In *SIGCOMM '90*, pages 209–219, September 1990.

[12] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The Digital Distributed System Security Architecture. In *Proceedings of the 12th National Computer Security Conference*, pages 305–319, 1989.

[13] Morrie Gasser and Ellen McDermott. An Architecture for Practical Delegation in a Distributed System. In *Proceedings of the Symposium on Security and Privacy*, pages 20–30, 1990.

[14] R. Govindan and D. Anderson. Scheduling and IPC Mechanisms for Continuous Media. In *ACM*, pages 68–79, 1991.

[15] G. Hamilton, M. L. Powell, and J. J. Mitchell. Subcontract: A flexible base for distributed programming. *Fourteenth Symposium on Operating Systems Principles*, pages 69–79, Dec 1993.

[16] Norman Hutchinson and Larry Peterson. The x-kernel: An archtecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–75, January 1991.

[17] E. A. Hyden. *Operating System Support for Quality of Service*. PhD thesis, Wolfson College, University of Cambridge, February 1994.

[18] N. Islam. *Customized Message Passing and Scheduling for Parallel and Distributed Applications*. PhD thesis, University of Illinois at Urbana-Champaign, 1994.

[19] Nayeem Islam, Robert E. McGrath, and Roy Campbell. "Parallel Distributed Application Performance and Message Passing: A case study". In *Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*, San Diego, California, September 1993.

[20] Butler Lampson. Protection. *ACM Operating Systems Review*, 8(1):18–24, January 1974.

[21] C. Maeda and B. N. Bershad. Networking Performance for Microkernels. In *Thirteenth ACM Symposium on Operating Systems Principles*, pages 154–159, April 1992.

[22] J. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Massachusetts, 1994.

[23] Richard Rashid. Threads of a New System. *UNIX Review*, 1986.

[24] Dennis M. Ritchie and Kenneth Thompson. The UNIX Time-Sharing System. *AT&T Bell Laboratories Technical Journal*, 57(6):1905, 1975.

[25] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocols: An Approach to Real Time Synchronization. *IEEE Transactions on Computers*, September 1990.

[26] Theron Tock, Daniel Sturman, and Roy Campbell. Security, Delegation, and Extensibility. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, August 1994.

[27] H. Tokuda, Y. Tobe, S. Chou, and J. Moura. Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network. In *COMM '92*, pages 88–98, 1992.