(2)

IDA PAPER P-2142

STRATEGIC DEFENSE SYSTEM DISTRIBUTED OPERATING SYSTEM R&D REVIEW AND RECOMMENDATIONS

Karen D. Gordon
Cathy J. Linn

April 1989

Prepared for
Strategic Defense Initiative Organization

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified		1b RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY		3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, unlimited distribution.	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE			
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Paper P-2142		5 MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses	6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION OUSDA, DIMO	
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311		7b ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	
8a NAME OF FUNDING/SPONSORING ORGANIZATION SDIO	8b OFFICE SYMBOL (if applicable) SDIO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031	
8c ADDRESS (City, State, and Zip Code) Room 1E149 The Pentagon Washington, DC 20301-7100		10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-R2-597.2 / WORK UNIT ACCESSION NO.	
11 TITLE (Include Security Classification) Strategic Defense System Distributed Operating System R&D: Review and Recommendations (U)			
12 PERSONAL AUTHOR(S) Karen D. Gordon, Cathy J. Linn			
13a TYPE OF REPORT Final	13b TIME COVERED FROM ______ TO ______	14 DATE OF REPORT (Year, Month, Day) 1989 April	15 PAGE COUNT 78
16 SUPPLEMENTARY NOTATION			
17 COSATI CODES FIELD / GROUP / SUB-GROUP		18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Distributed operating systems; real-time computing; reliability; fault tolerance; security; Alpha Distributed Operating System; Cronus Distributed Operating System (Continued)	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The Strategic Defense System (SDS) imposes a set of requirements on distributed operating systems that is not met by state-of-the-art systems. In this paper, the key requirements are identified as being real-time support, reliability/fault tolerance, and security. The extent to which these requirements are being addressed by current distributed operating system research is discussed. The three distributed operating system projects that are currently receiving SDIO funds—Alpha, Cronus, and Mach—are reviewed. A fourth project, the V distributed system project of Stanford University, is also highlighted, because of its unique potential for meeting certain SDS needs. Recommendations on the directions in which the SDIO should pursue each of these projects are made. (Continued)

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Cathy Jo Linn	22b TELEPHONE (Include area code) (703) 824-5520	22c OFFICE SYMBOL IDA/CSED

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

18. SUBJECT TERMS (Continued)

Mach Distributed Operating System; V Distributed Operating System; Amoeba Distributed Operating System; Clouds Distributed Operating System; Heterogeneous Computer Systems (HCS).

19. ABSTRACT (Continued)

The Office of Naval Research (ONR) Real-Time Computing Initiative, which is addressing some issues critical to the development of the SDS, is described. It is recommended that the SDIO seek to coordinate with the ONR in this effort.

The appendix to this paper provides detailed descriptions of the Alpha, Cronus, Mach, and V distributed operating system projects, as well as of three other projects noted in the body of the paper: Amoeba, Clouds, and the Heterogeneous Computer Systems (HCS) Project.

UNCLASSIFIED

IDA PAPER P-2142

STRATEGIC DEFENSE SYSTEM DISTRIBUTED OPERATING SYSTEM R&D REVIEW AND RECOMMENDATIONS

Karen D. Gordon
Cathy J. Linn

April 1989

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-R2-597.2

UNCLASSIFIED

Preface

This document, IDA Paper P-2142, was prepared for the Strategic Defense Initiative Organization (SDIO) in response to a request to review its distributed operating system research and development (R&D) program.

The Strategic Defense System (SDS) imposes a set of requirements on distributed operating systems that is not met by state-of-the-art systems. In this paper, the key requirements are identified as being real-time support, reliability/fault tolerance, and security. The extent to which these requirements are being addressed by current distributed operating system research is discussed.

The three distributed operating system projects that are currently receiving SDIO funds - Alpha, Cronus, and Mach - are reviewed. A fourth project, the V distributed system project of Stanford University, is also highlighted, because of its unique potential for meeting certain SDS needs. Recommendations on the directions in which the SDIO should pursue each of these projects are made.

The Office of Naval Research (ONR) Real-Time Computing Initiative, which is addressing some issues critical to the development of the SDS, is described. It is recommended that the SDIO seek to coordinate with the ONR in this effort.

The appendix to this paper provides detailed descriptions of the Alpha, Cronus, Mach, and V distributed operating system projects, as well as of three other projects noted in the body of the paper: Amoeba, Clouds, and the Heterogeneous Computer Systems (HCS) Project.

It should be pointed out that this document was prepared in two parts, the body of the paper and the appendix, to facilitate its distribution. In particular, it is envisioned that the appendix, which offers uncritical overviews of the distributed operating system projects, will have a wider distribution than the body of the paper, which may be judged to contain some sensitive material.

Acknowledgments

The investigation reported herein was undertaken at the initiative of LTC Jon Rindt of the SDIO. He and Lt Col Chuck Lillie recognize the critical role that distributed operating systems will play in the SDS, as well as the significant challenge that the SDS poses for distributed operating system researchers and developers. The authors thank both of them for their support of this effort.

The authors gratefully acknowledge the cooperation and assistance received from the sponsors, principal investigators, and staff of the operating system projects reviewed in this paper. The following individuals provided excellent overviews of, and insight into, their respective projects, in the form of briefings, literature, informal discussions, and reviews of this document:

- Alpha: Doug Jensen, Concurrent Computer Corporation; Duane Northcutt and Ray Clark, Carnegie Mellon University; Dick Metzger and Tom Lawrence, Rome Air Development Center
- Clouds: Rich LeBlanc, Partha Dasgupta, and Ray Chen, Georgia Institute of Technology
- Cronus: Steve Vinter and Rick Schantz, BBN Laboratories; Dick Metzger, Tom Lawrence, Rome Air Development Center
- Mach: Rick Rashid, Carnegie Mellon University
- ONR Real-Time Computing Initiative: Andre van Tilborg, Office of Naval Research
- V: David Cheriton, Stanford University

The authors also gratefully acknowledge the support received from Anand Tripathi of the University of Minnesota. He prepared the initial draft of the Amoeba summary and carefully reviewed the other project summaries.

Finally, the authors thank the IDA/CSED reviewers of this document for their many helpful suggestions. Terry Mayfield, in particular, provided an exceptionally thorough and thoughtful review.

Table of Contents

1. INTRODUCTION

The Strategic Defense Initiative Organization (SDIO) recognizes the significant role that distributed operating systems will play in the Strategic Defense System (SDS). It is currently supporting three major distributed operating system projects: Alpha, Cronus, and Mach. Alpha is being developed at Carnegie Mellon University, under the leadership of Doug Jensen.[1] The niche that it claims for itself is real-time computing. Cronus is being developed at BBN Laboratories.[2] Its niche is the integration of computer systems that are heterogeneous with respect to hardware, as well as software. In effect, Cronus is a "meta-" operating system. It resides on top of native operating systems such as UNIX[3] or VMS[4], essentially providing a layer (or layers) of coherence and uniformity between distributed applications and the underlying centralized operating systems. Mach is also being developed at Carnegie Mellon University, under the leadership of Rick Rashid. As a UNIX-compatible operating system, its niche is the extension of UNIX functionality to advanced architectures and environments, such as those encompassing multiprocessors, large memories, and high-performance networks.

The SDIO supports these distributed operating system projects by funding Work Package Directives (WPDs) that are executed by other Department of Defense services and agencies. The services and agencies, in effect, serve as the sponsors of the projects. Both Alpha and Cronus are sponsored by the Air Force through its Rome Air Development Center (RADC), and Mach is sponsored by the Defense Advanced Projects Research Agency (DARPA). In each case, the sponsoring organization is dedicated to the project, maintains a strong relationship with the principal investigator(s), and has played a major role in the development of the system.

Each of the three projects was conceived independently of the SDS and its requirements. While all the projects have since been cast, to varying degrees, in SDS terms, their basic goals remain unchanged. The SDIO is concerned about how well those goals respond to SDS requirements, and how well the projects are doing in achieving the goals.

To obtain an independent assessment of its distributed operating system research and development (R&D) program, the SDIO requested the Institute for Defense Analyses (IDA) to critically review its three distributed operating system projects – Alpha, Cronus, and Mach – and furthermore to make recommendations on future SDS distributed operating system R&D. The review and recommendations were to be made in light of SDS requirements and also in light of other R&D in the field of distributed operating systems. This paper represents IDA's formal response to that SDIO request.[5]

Section 2 of this paper identifies the requirements of distributed operating systems for the SDS. Section 3 reviews the three SDIO-funded distributed operating system projects, as well as the V distributed system project, which is put forward in this paper as holding unique potential for contributing to SDS development. Section 4 describes the Office of Naval Research (ONR) Real-Time Computing Initiative, which is another effort deemed to hold promise in meeting some of the most challenging SDS requirements. Finally, Section 5 presents our recommendations on the directions in which the SDIO should pursue distributed operating system R&D.

The appendix to this paper provides detailed descriptions of the four distributed operating systems (i.e., Alpha, Cronus, Mach, and V) that are the primary subjects of the paper. It

[1] Jensen is no longer at Carnegie Mellon University; he is now at Concurrent Computer Corporation. However, he continues to direct the Alpha project.

[2] The Cronus project does not have a single long-term project leader, or Principal Investigator, in the sense that most university-based research efforts do.

[3] UNIX is a registered trademark of AT&T Bell Laboratories.

[4] VMS is a trademark of Digital Equipment Corporation.

[5] Recommendations have already been provided informally, through the delivery of drafts of this paper to the SDIO SDS Phase One Program Office and through participation in the Phase One Engineering Team (POET) Software Engineering and Operating Systems Panel.

also provides descriptions of the Amoeba, Clouds, and Heterogeneous Computer Systems (HCS) projects. The latter three projects are covered for comparison purposes. Besides being among the most prominent ongoing distributed operating system efforts, each is referred to in the body of the paper as being a competitor, in certain aspects, of one of the four key projects. (In particular, Amoeba, like V, emphasizes performance and minimization of the kernel; Clouds, like Alpha, utilizes the object/thread programming model; and HCS, like Cronus, seeks to integrate systems that are heterogeneous with respect to both hardware and software.)

2. SDS DISTRIBUTED OPERATING SYSTEM REQUIREMENTS

The SDS and other large, complex, mission-critical, real-time systems demand that the scope of distributed operating system research be broadened beyond the interactive computing environment, which has been the focus of much of the research up to this point in time. These systems pose requirements that challenge the state of the art in distributed operating system technology in three key areas: (1) real-time computing, (2) reliability and fault tolerance, and (3) security.

Before proceeding with a discussion of the challenges and requirements in each of these areas, let us stress that the SDS is not envisioned as a monolithic system under the the control of a single, universal distributed operating system that must meet the most stringent requirements in each of these three areas at once. At the highest level, it is seen as at least two systems: (1) the SDS development and maintenance system and (2) the deployed SDS system. In turn, the SDS development and maintenance system will consist of interconnected heterogeneous networks of heterogeneous systems. The individual systems will include both uniprocessor and multiprocessor configurations, some of which may run system-unique operating systems. The deployed SDS system will also consist of interconnected networks of uniprocessor/multiprocessor systems; however, the degree of heterogeneity will be controlled. These two systems – the SDS development and maintenance system and the deployed SDS system – place very different demands on distributed operating systems, in particular, on the real-time, reliability/fault tolerance, and security aspects of the systems.

It is not certain how far the boundaries of individual distributed operating systems can or should extend in the SDS. However, it can be presumed, given the state of the art, that the boundaries will extend far enough to encompass the computing resources of a local area network or of a space-based platform. Our review is based on this presumption.

2.1. Real-Time Computing

As noted above, current distributed operating system research is focused on interactive computing. Common goals are to minimize average delay (e.g., response time) or to maximize average throughput. "Fairness" is also important. In short, interactive computing systems are designed to be responsive to their human users.

Real-time computing systems, on the other hand, must be responsive to the total environment in which they exist. This environment may consist of a larger system (e.g., automobile, aircraft, factory floor, nuclear power plant, etc.), which may in turn be intimately linked to its physical environment (e.g., temperature, humidity, road conditions, air turbulence, threats, etc.). The total environment in which a real-time computing system is embedded imposes timing constraints on the system. The timing constraints are often in the form of the hard deadlines that if not met can lead to catastrophic failures. Such timing constraints distinguish real-time systems from other systems.

Recently, the proliferation of real-time computing systems has heightened interest in real-time computing [Stankovic 88]. The problem is that today's real-time systems are built through *ad hoc* techniques and at inordinate expense. As systems become ever larger and more complex, a more disciplined approach becomes essential. In his article, Stankovic encourages the community of researchers, developers, and funding agencies to work together in developing a scientific foundation for real-time computing. The ONR Real-Time

Computing Initiative, which is discussed in Section 4 of this paper, has been launched with the objective of building such a scientific foundation.

Research Challenges

The basic challenge for distributed operating system researchers, then, is to move beyond interactive applications to real-time applications, especially large, complex ones. In other words, the challenge is to contribute to the laying of a scientific foundation for real-time computing, by providing a real-time distributed operating system upon which real-time applications can be built.

Traditionally, real-time computing has been associated with speed. While speed remains a pressing issue, the real-time systems research community has expanded the scope of real-time computing to encompass the notion of "predictability." There are two aspects to this notion: one dealing with *correctness*, and the other with *methods of assurance*. First, in regard to correctness, timeliness is elevated to a first-order system concern; it is considered to be a mandatory aspect of correctness, along with functional or logical correctness. A system that produces correct outputs, but fails to meet timing constraints, is not correct. Second, in regard to methods of assurance, the goal is to eliminate the need for exhaustive testing. The idea is to develop "constructive" methodologies, namely, ones that yield predictable results. Assurance is obtained through *a priori* proofs about systems constructed via the methodology, rather than through exhaustive testing of a completed system. Rate-monotonic scheduling, for example, serves as the basis for several new scheduling algorithms [Lehoczky 87] [Sha 87] [Sha 88] which (under specified conditions) guarantee that all deadlines will be met.

Specific issues for real-time distributed operating systems include the following:

- Fast response: Proponents of predictability tend to dismiss fast response as implying nothing more than raw speed, which can be obtained through hardware technology. However, fast response entails more than processor speed; it demands an effective operating system design. Claiming that fast response is insignificant is like claiming that "fast" algorithms are insignificant. The difference is that schemes for quantifying the performance of algorithms *in processor-speed-independent terms* have been devised. These schemes are based on the number of "steps" per input and allow us to classify algorithms in the familiar O-notation (e.g., $O(\ln n)$, $O(n)$, $O(n^2)$, etc.). Unfortunately, it is not so easy to characterize the performance of operating systems with a single metric (or even a few metrics) in this way, but this does not obviate the need for performance-conscious system design.
- Consistent management of *all* system resources: Real-time support must extend beyond single processors, both to multiprocessing and distributed processing architectures, and to other shared logical and physical resources. Priority inversion, in which a lower priority task blocks a higher priority task, must be minimized and accounted for.
- Accommodation of aperiodic tasks: Real-time support must extend to aperiodic tasks, as well as to periodic ones.
- Accommodation of widely varying time scales: Real-time support must provide for the integrated management of tasks whose time scales vary from microseconds or less (e.g., sensor input processing) to minutes or more (e.g., global planning and resource management).
- Robustness: Real-time support must provide for stability under transient overload. It must ensure that the least "important" tasks are the ones that miss their deadlines (or otherwise suffer) first.
- Fault tolerance: Real-time support cannot assume a failure-free environment. It must enable system operation and performance to degrade gracefully as components fail.

SDS Requirements

Real-time response is a fundamental requirement of distributed operating systems for the deployed SDS. "Functional" or "logical" correctness of a result without timeliness of the result serves no purpose; in fact, the system that produces the result is "incorrect" if it does not meet timing constraints. Furthermore, as a system of incomparable size and complexity, the SDS simply cannot rely on current *ad hoc* development techniques; it epitomizes the need for a more disciplined approach to achieving real-time computing.

In contrast to the deployed SDS, the SDS development and maintenance system does not call for real-time response as a mandatory requirement. It is not safety-critical in the same sense as the deployed SDS; hence, its performance requirements are not as stringent as those of the deployed SDS.

2.2. Reliability and Fault Tolerance

System reliability can be increased in two basic ways: (1) fault avoidance, i.e., increasing the reliability of components and applying conservative design practices and (2) fault tolerance, i.e., employing redundancy to achieve tolerance to the component failures that cannot ever be totally eliminated. The focus here is on fault tolerance, because it is the area in which the operating system, as the system resource manager and controller, has a role to play.

At the highest level, fault tolerance can be viewed as addressing two distinct but interrelated concerns: data integrity and processing integrity. Data integrity, in this context, deals with ensuring that data can survive component failures – that data is not lost, corrupted, or made inconsistent. Processing integrity deals with ensuring correct and continuous processing in the face of component failures.

The importance of each of these aspects of fault tolerance is environment, or application, dependent. Data integrity is crucial for long-lived data. Consequently, data integrity is closely linked to database system technology, and has been much explored in that domain; in fact, many of the mechanisms (e.g., data replication, atomic transactions) utilized by distributed operating systems originated in the context of database systems. Processing integrity, on the other hand, is most crucial for real-time safety-critical operations.

Current distributed operating system research, with its emphasis on interactive computing, has been focused on data integrity. Mechanisms such as data replication, atomic transactions [Gray 79], and nested transactions [Moss 81] continue to be explored in great depth [BBN 88a] [BBN 88b] [Dasgupta 88] [Liskov 88] [Specter and Swedlow 88] [Tripathi 88].

On the other hand, processing integrity, in the form of continuous (or non-stop) processing, is not perceived as being critical in the interactive computing environment, and it has not received much attention from the distributed operating system research community. There is, however, a substantial body of research devoted to this aspect of fault tolerance, stemming from the requirements of aerospace applications, telephone switching systems, industrial process control applications, and on-line transaction processing services [Nelson and Carroll 87] [Rennels 84] [Serlin 84]. Hardware redundancy is the basis of fault tolerance in these applications.

Research Challenges

The standard models for achieving fault tolerance are simply too costly, in terms of money and/or time, to be used routinely. Triple-modular redundancy, for example, demands extraordinary hardware replication; atomic transactions pay significant performance penalties in terms of both delay and availability.

Therefore, the challenge for distributed operating system researchers is three-fold: first, to develop more cost-effective and performance-effective mechanisms for achieving fault tolerance; second, recognizing that the implementation of fault tolerance will always incur some cost and/or performance penalties, to develop tailorable mechanisms that can achieve *varying* levels of fault tolerance for *varying* costs (in terms of money as well as performance); and, third, to capitalize on, as well as improve upon, the results that have been achieved in the domain of fault-tolerant computer architectures. Architecture mechanisms (e.g., sparing, backup, triple-modular redundancy) and distributed operating system mechanisms (e.g., data replication, atomic transactions, nested transactions) must be made to work together, yielding a unified approach to fault tolerance.

SDS Requirements

In terms of reliability and fault tolerance requirements, the SDS development and maintenance system is no different from any other development and maintenance system. That is, as a system expected to hold long-lived data, it has an inherent requirement for data integrity. In regard to processing integrity, the SDS development and maintenance system does not have extraordinary requirements. Thus, the massive hardware redundancy often employed in real-time safety-critical systems to detect processing faults and to ensure continuous operation is not demanded. The SDS development and maintenance system can tolerate temporary interrupts or delays in processing capability without suffering unduly adverse consequences.

To consider the reliability and fault tolerance requirements of the deployed SDS, let us focus on two aspects: (1) ground-based command and control and (2) space-based operations. The ground-based command and control functions place high demands on both data integrity and processing integrity; correctness and continuity of both data and processing is vital. The ground-based components will undoubtedly employ sufficient redundancy to ensure that the integrity demands are met. Space-based platforms, on the other hand, cannot afford the redundancy that is required to achieve ultrahigh levels of data and processing integrity. They have to be very judicious in applying redundancy, which can lead to either underutilized resources or excessively large time delays in a distributed system such as the SDS.[6] Their goal is to "optimally" apply all available resources in performing their mission. Hence, the space-based platforms demand tailorable fault tolerance support, which allows tradeoffs to be made between fault tolerance and performance.

Beyond fault tolerance lies the concept of graceful degradation. The SDS must not only incorporate fault tolerance mechanisms, it must also be designed to degrade gracefully. The redundancy employed to achieve (some degree of) fault tolerance can only go so far; it can protect against single points of failure or limited occurrences of multiple failures. But in a hostile environment such as that in which the SDS will exist, failures will eventually exceed the capacity of the system (through its built-in redundancy) to withstand them. When this happens, the system cannot collapse; it must degrade gracefully. That is, it must continue to operate, albeit in degraded states. Moreover, the degradation should be in some sense commensurate with the extent of damage to the system. The SDS must be designed with graceful degradation in mind. The highest levels of design, such as the overall system architecture and the battle management strategy, are especially crucial in this regard. The operating system, through its dynamic management of resources, provides support for graceful degradation.

2.3. Security

Security can be viewed as encompassing three aspects: (1) confidentiality, (2) integrity, and (3) availability. The first aspect, confidentiality, is the focus of this subsection. It should be emphasized that in focusing on confidentiality, we are not implying that the other aspects of security (integrity and availability) are any less important. We are simply reflecting the fact that confidentiality is better understood. While many issues remain, the confidentiality aspect is sufficiently developed so that it can be productively investigated in the context of distributed

[6] It was these considerations that led, in part, to the current SDS concept of a decentralized architecture with implicit coordination among components of the architecture. The decentralized architecture avoids the need for absolute data consistency on a global scale. The absolute consistency (on the global scale) is traded for enhanced timeliness and availability of data (on the local scale).

operating systems. The integrity[7] and availability aspects, on the other hand, fall outside the scope of distributed operating system research, at least at this time. They remain to be investigated and further developed by the security research community.

To elaborate on the security challenge for distributed operating system researchers, let us first review current security practices, and then discuss security as defined in the National Computer Security Center (NCSC) Trusted Computer Systems Evaluation Criteria (TCSEC) (commonly referred to as the "Orange Book") [TCSEC 85], the authoritative Department of Defense reference on computer system security. It should be noted that the TCSEC primarily addresses confidentiality.

Prevailing Practice

The purpose of distributed operating systems is to facilitate the logical unification of physicaly interconnected computing resources. Unification of resources entails both the enabling and the constraining of resource sharing, for both physical resources and information resources. Distributed operating systems must manage and control resource sharing, just as centralized systems must. They must protect potential "sharers" from one another.

In distributed systems, resource sharing is implemented primarily through interprocess communication. Subjects (e.g., user processes) and objects (or, in some cases, object managers) are bound to private address spaces, whose integrity is ensured through virtual memory concepts and mechanisms. Subjects access objects through interprocess communication. Distributed operating systems provide security, or *protection*, by constraining interprocess communication.

Interprocess communication is constrained by maintaining and utilizing information on *which* subjects can perform *which* operations on *which* objects. This information is maintained in the form of capabilities or access control lists. Capability systems associate security information with subjects (invokers), whereas access control list systems associate the information with objects or object managers (invokees). For example, in an access control list system, each file has associated with it a list of authorized users (and uses). In a capability system, each subject has a "capability" for each object it may access. A capability serves as a globally unique object name and as a "ticket" for access. In its role as a ticket, the capability confers the access rights specified in the capability upon each subject possessing the capability.

TCSEC

The TCSEC discusses access control policy, as well as measures for achieving accountability and assurance with respect to the implementation of the policy. It distinguishes two types of access control: mandatory access control (MAC) and discretionary access control (DAC). Prevailing distributed operating system protection practices are more closely related to DAC than to MAC.

In DAC, access to an object is controlled, at the discretion of the owner of the object, solely on the basis of user identity. That is, access is restricted based on the identity of the user making the access, or on whose behalf the access is being made. Thus, the fundamental requirement of distributed operating system kernels is to provide secure user authentication. Then, either the kernel or servers can implement DAC. Kernel implementation offers uniformity; server implementation offers flexibility.

[7] It should be noted that (data) integrity, in the context of security, has to do with the protection of data against alteration by unauthorized users. In other words, the threat to data integrity that is of primary concern in the security context is unauthorized users, whereas in the previous section the primary threat to data integrity was component failures. It should be further noted that discretionary access control addresses data integrity to some extent, by, for example, restricting the set of subjects having write access to a given object.

Access control lists are the security mechanism envisioned by the TCSEC. They provide a straightforward implementation of DAC. Traditional capability systems fail to provide DAC (as defined in the TCSEC) for several reasons [Branstad 88] [Gligor 87] [TIS 88]. The fundamental problems are that (1) access is controlled by ticket (i.e., capability) possession rather than user identity, (2) a ticket can be passed on to other subjects by any holder of the ticket, and (3) an object does not maintain records of which subjects have been granted access to it. However, traditional capability systems can be enhanced or modified to comply with TCSEC requirements [Branstad 88] [Gligor 87] [TIS 88].

In MAC, subjects are granted access on the basis of both user identity and clearance for access to information of specified "sensitivity levels." Subjects and objects are labeled with sensitivity levels, and access control conforms to a specified security model. MAC is designed to reflect traditional manual security practices, in which people are assigned clearance levels (e.g., unclassified, confidential, secret, top secret) and documents are assigned corresponding sensitivity levels. It is also designed to provide mandatory control of information flow, rather than just access. A system "containing information with different sensitivities that simultaneously permits access by users with different security clearances and needs-to-know, but prevents users from obtaining access to information for which they lack authorization" is referred to as a *multilevel-secure* system [TCSEC 85, p. 114].

Research Challenges

Providing MAC, DAC, and associated accountability and assurance measures as defined in the TCSEC is a challenge, even in centralized systems. Few systems that meet the most stringent requirements of the TCSEC (namely, A1 certification) have been built. Moreover, the TCSEC was written with centralized systems in mind; applying its concepts and requirements to other systems is a difficult task. An interpretation for networks, known as the Trusted Network Interpretation (TNI) or "Red Book", has been prepared [TNI 87]. A Trusted Database System Interpretation (TDI) is also being developed.

Distributed systems are indeed a challenge for TCSEC interpretation. They complicate the provision of security in the following ways:

- Decentralized control: Distributed systems attempt to provide a level of resource unification commensurate with that of centralized systems. But they carefully avoid centralized control. So, instead of having one mediator for access control, distributed operating systems have multiple mediators, one in each kernel. The coordination of these mediators is the challenge here.
- Untrusted kernels: In some network environments, for example ones in which personal computers predominate, it may not be possible or desirable to rely on the trustworthiness of kernels. Malicious users could tamper with kernels. Security mechanisms that enable untrusted kernels to participate in trusted distributed systems are the challenge in this case [Tanenbaum and van Renesse 85].
- Network communication: Communication over an underlying network introduces security threats that must be taken into account. These include peeking, impostoring, message tampering, and replays. Encryption, coupled with other mechanisms such as transaction numbering, can be used to counter these threats [Cheriton 88a].

Despite these complications, all is not gloomy with respect to security. Many distributed operating systems have design features that actually facilitate the implementation of security. Among these are the following:

- Minimal kernel: Many distributed operating systems are designed according to the minimal kernel philosophy, which espouses minimization of code that runs in kernel mode. A minimal kernel typically provides processes and interprocess communication; application-level servers then provide higher-level system objects and services (most notably, a file system). The minimal kernel serves as the focal point for implementation of security and is, in fact, required by the TCSEC.

- Interprocess communication: Distributed operating systems tend to be communication-centric. Resource sharing is accomplished via interprocess communication. By managing and controlling interprocess communication, distributed operating systems can provide secure sharing of the resources under its control. Thus, interprocess communication further focuses security concerns.
- Global user identification: In supporting network transparency, distributed operating systems implement global user identification schemes. Since user identification is fundamental to access control, having global schemes facilitates its implementation.
- Global object naming: Distributed operating systems also implement global (uniform) object naming schemes. Such schemes serve as a basis for implementation of both discretionary and mandatory access control.

As in the case of reliability and fault tolerance, the ultimate challenge in regard to security is to provide *varying* levels for *varying* costs. For example, it may be desirable for trusted nodes (kernels) to interoperate with untrusted nodes. Likewise, it may be desirable for trusted and untrusted servers to interoperate. Or, different applications may require different levels of assurance of security, which can manifest themselves as, for example, different protection algorithms or different methods of program verification. The goal is to be able to make tradeoffs between level of security and cost in terms of money or performance.

SDS Requirements

Given the state of the art in the security area, it is not clear how far the TCSEC concepts and requirements can or should extend in the SDS. Clearly, traditional protection in the form of DAC is essential. But, while multilevel-secure distributed operating systems (which implement MAC) are often postulated and would offer certain advantages, they are not deemed to be essential. Alternatives to multilevel-secure operation do exist. The alternatives generally rely on one of two measures (or a combination of both):

- Restricting the user population (through the system-high mode of operation).
- Physically isolating information of different security levels.

In system-high mode, all data or objects in a system are treated as if they were classified at the highest level allowed in the system. Only subjects cleared to the system-high security level can access (e.g., read or write) *any* of the objects in the system. For example, in a Top Secret system-high facility, objects of Unclassified, Confidential, Secret, or Top Secret classification can be stored, but only Top-Secret-cleared subjects can access any of the data.[8] Authentication and access control can be implemented by physical measures (e.g., guarded vaults) or by automated measures (in accordance with the Orange Book [TCSEC 85]).

System-high operation seems to be a reasonable mode of operation for the deployed SDS. That is, its chief drawback, the restriction of the user population, does not seem to be too stiff a penalty to pay for security. It has, in fact, been suggested as the preferred approach in [SPARTA 88], largely because of concerns that the implementation of multilevel security could impose significant (possibly intolerable) performance penalties on real-time response.

While multilevel-secure distributed operating systems may not have a role to play in the deployed SDS, with its emphasis on real-time response, they do appear to be more promising and more desirable in the SDS development and maintenance environment. In this environment, data of all security levels will exist, and the number of personnel involved will be large. Global system-high operation is undesirable and probably infeasible. Penalties would be incurred in one of two ways. Either there would have to be an inordinately large number of

[8] In a typical multilevel-secure system, Confidential subjects could read from Unclassified and Confidential objects and write to objects of Confidential or higher classification; Secret subjects could read from Unclassified, Confidential, and Secret objects and write to objects of Secret or higher classification; etc. In other words, subjects could read objects of equal or lower sensitivity and write objects of equal or greater sensitivity; "read-up" and "write-down" operations would be disallowed.

(high) personnel clearances; or some qualified personnel would end up being denied access, and their expertise would be lost.

Since global system-high operation is undesirable for the SDS development and maintenance environment, it is necessary to resort to the other security measure, some degree of physical isolation of information of different security levels. For example, there could be a Secret system and a Top Secret system, each operated in (local) system-high mode. Isolation occurs in the sense that information of a given security level is physically isolated from systems to which users of lower clearance levels have access; i.e., the Top Secret information contained in the Top Secret system cannot be accessed by Secret (or Confidential or Unclassified) subjects. This solution sacrifices unification of resources. For example, the same Secret data might appear in both systems. Maintaining consistency in such a case would be at best cumbersome. If the systems were run in dedicated mode, wherein a system is dedicated to containing information of a single security level, then users, or subjects, must access multiple systems, which is also undesirable. For example, if the Secret data did not exist in both systems, then a Top Secret subject would be forced to access both systems.

Other alternatives for the SDS development and maintenance system, which focus on providing unification of specific resources include the following: (1) separate systems for different security levels (run in either local system-high mode or dedicated mode), interconnected with multilevel-secure communication networks, which provide for unification of communication resources [Abrams and Podell 87]; and (2) multilevel-secure data management (for example, via the integrity lock approach [Denning 84] [Graubart 84], which utilizes cryptographic checksums), which provides for unification of data.

Finally, it should be noted that there will need to be a link between the SDS development and maintenance system and the deployed SDS. The link will provide the means for modifying software in the deployed SDS. The security of the link is vital. This issue has been proposed for further investigation by RADC [RADC 88].

3. REVIEW OF SPECIFIC DISTRIBUTED OPERATING SYSTEM PROJECTS

This section critically reviews the three SDIO-funded distributed operating system projects: Alpha, Cronus, and Mach. In addition, the V distributed system project of Stanford University, which is sponsored in part by DARPA, is reviewed, because of its well-established position in the distributed operating systems field and its potential for meeting SDS requirements.

The reader is referred to the Appendix for detailed descriptions of the four distributed operating system projects reviewed herein. Therein, each description addresses project goals, technical approach, and current status. Additionally, each description provides an extensive list of references.

3.1. Alpha

Target Domain

Of all the distributed operating system efforts, Alpha is the one whose stated goals most closely match the requirements of the deployed SDS. Alpha's target domain is distributed, real-time Battle Management/Command, Control, and Communication (BM/C3) systems. Therefore, in SDS terms, its target domain is the deployed SDS, as opposed to the SDS development and maintenance system.

In claiming distributed, real-time BM/C3 systems as its target domain, Alpha is implicitly assuming responsibility for addressing real-time, reliability/fault tolerance, and security issues. Its uniqueness lies in the real-time area; it is devoted to time-driven resource management. Innovations in the other areas are not apparent. In regard to reliability and fault tolerance, mechanisms based on replication and atomic transactions are being pursued in the context of Ph.D. thesis research, but results have not been reported. In regard to security, Alpha utilizes capabilities in the traditional way for both naming and protection. The general design of the Alpha kernel (i.e., the basic abstractions of object, thread, and object invocation[9]) was adopted from the Clouds distributed operating system of the Georgia Institute of Technology.[10] Utilizing the same general design, the projects emphasize different research topics. Emphasis in the Alpha project is on real-time computing, whereas emphasis in the Clouds project is on reliability and fault tolerance.

Status

Alpha is in an early stage of development. Work has concentrated on the kernel; system services have not been implemented, and programming support is minimal. In regard to the kernel, time-driven resource management has been limited to uniprocessor scheduling. Reliability and fault tolerance, which are viewed as being vital aspects of the Alpha kernel, is still in the process of being designed and implemented. Security work has been limited to capability-based protection.

Concerns

The concept of time-driven resource management was formulated by Doug Jensen, Doug Locke, and Hide Tokuda [Jensen 85] [Locke 86] in the context of Carnegie Mellon University's Archons Project (which dates back to 1979 and of which Alpha is the operating system effort). In time-driven resource management, both the time constraints and the relative importance of each computation (in this case, thread) are specified, by a time-value function. Time-value functions, in conjunction with best-effort scheduling, are notable in that they offer a means to deal with (1) aperiodic tasks, (2) transient overload (which is bound to occur under stress conditions, when it is actually most important for the system to perform its mission), and (3) soft deadlines.

While time-driven resource management is an intuitively appealing approach to real-time resource management, its general viability and its applicability to the SDS remain unproven. Time-driven resource management is computationally expensive. Satisfactory techniques for the assignment of importance values to tasks have not been formulated. Further research is needed to convincingly demonstrate the applicability of time-driven resource management to SDS problem domains.

Although the Alpha distributed operating system is sometimes presented (in Alpha briefings) as being uniquely suitable for the investigation and implementation of time-driven resource management, it is in fact not so. Time-driven resource management is also being investigated in the context of Mach [Tokuda 87].

As previously noted, time-value functions and best-effort scheduling offer means of dealing with aperiodic tasks and transient overloads, both of which must be handled in the deployed SDS. However, other approaches do exist, and are being pursued in earnest in the context of the ONR Real-Time Computing Initiative, which is described in Section 4. Some, for example, are based on fixed-priority, rate-monotonic scheduling. They incorporate extensions for both aperiodic tasks and transient overloads. Furthermore, they address the synchronization of shared resources and the resulting potential for priority inversion.

The goals of the Alpha project are indeed well articulated, and, furthermore, align with many of the goals of the SDS. However, in the view of the distributed operating system research community, the accomplishments of the project are more modest. Publications are limited; at the same time, much system development remains to be done, despite the number of years that have been spent in the problem domain.

[9] Descriptions of these abstractions are provided in the Appendix.

[10] A description of the Clouds project is provided in the Appendix.

3.2. Cronus

Target Domain

The intent of the Cronus project is to support interactive users in building and running large-scale applications in a distributed computing environment marked by heterogeneity – of architectures, operating systems, and other software. With respect to the SDS, RADC and BBN have cited the SDS development and maintenance system as the target domain of Cronus. Cronus is not suited for real-time systems such as the deployed SDS.

In regard to reliability and fault tolerance, Cronus addresses data integrity via replication and atomic transactions. Notably, the Cronus project has recently moved toward providing some flexibility/tailorability in its support of replication. This enables application developers to trade consistency for availability.

In regard to security, Cronus utilizes access control lists to provide discretionary protection. Additionally, multilevel security was addressed in a related project, as discussed below under *Concerns*.

Status

The Cronus project has proven its concept of integrating heterogeneous computer systems by imposing a layer of standardization (in the form of the Cronus distributed operating system) on top of (heterogeneous) native operating systems. The Cronus system is a mature system.

Concerns

The Cronus system has not been widely applied in the general community (i.e., outside of RADC, BBN, and MITRE). Its effectiveness in supporting distributed application development remains to be convincingly demonstrated.

Since the initiation of the Cronus project in 1981, much progress has been made in the area of integrating heterogeneous computer systems. Among the most promising approaches that have been developed are the following: 1) evolving International Organization for Standardization (ISO) Open Systems Interconnection (OSI) standards, which are addressing distributed application development, and represent *international* standards for distributed application development; 2) Heterogeneous Computer Systems (HCS) Project of the University of Washington,[11] which is relying on emulation and accommodation of multiple standards rather than resorting to new standards; and 3) existing data communication protocols, which offer limited functionality (in the form of remote login, file transfer, and electronic mail), but which are (almost) universally implemented and may suffice in the short term. Each of these approaches enjoys a broader base of interest and support than the Cronus approach.

In a sense, portable distributed operating systems (that provide their own native kernels) are also competitors of Cronus. They enable diverse hardware architectures to be integrated into distributed systems. Some (e.g., Amoeba, Clouds, V) have felt compelled to emulate the UNIX programming interface, to varying degrees, so they can host the abundant supply of UNIX-based software and be appealing to large numbers of programmers. Native-kernel distributed operating systems can emulate other operating system programming interfaces as well. They can provide a higher degree of resource unification than Cronus, which must rely on underlying centralized operating systems for some of its functionality. All of the native-kernel distributed operating systems covered in this paper (i.e., Alpha, Amoeba, Clouds, Mach, and V) have portability as a goal, in that they strive to minimize machine dependence of the operating system design and code.

[11] A description of the HCS Project is provided in the Appendix.

Now, let us move on to security considerations. In a Cronus-related RADC/BBN project, the Secure Distributed Operating System (SDOS) Project, the question of how to incorporate multilevel security into Cronus was investigated. The following conclusion was reached [Casey 87, p.19]: "Thus, the host operating system(s) on top of which SDOS [i.e., *secure* Cronus] is implemented must have a minimum of a B2 rating, and ratings of B3 or A1 are more desirable." GEMSOS, a product of Gemini Computers, Inc., of Carmel, California, was recommended as the multilevel-secure operating system upon which to base SDOS.

This recommendation (GEMSOS or other B3 or A1 operating systems as the native operating systems) seems antithetical to Cronus's chief purpose of integrating computer systems with *heterogeneous* operating systems. To achieve multilevel security, a *restricted* base of multilevel-secure native operating systems must be installed; heterogeneity, Cronus's overriding reason for being, is sacrificed in the pursuit of multilevel security. The implication is that the utility of Cronus in a multilevel-secure, heterogeneous computing environment (such as the SDS development and maintenance system) is limited.

3.3. Mach

Target Domain

Mach represents DARPA's effort at establishing an operating system standard for distributed computing environments. It capitalizes on UNIX's *de facto* standard status by having UNIX binary compatibility as a basic tenet. It extends UNIX functionality to advanced computing architectures and environments, namely, those encompassing multiprocessors, large memories, and high-performance networks.

Due in part to its UNIX heritage, the primary target domain for Mach is interactive computing. However, DARPA continues to expand its vision of Mach's role. Real-Time Mach, for example, is in preliminary stages of development. (That is, initial work has been reported in the literature [Tokuda 87], and Mach briefings often cite the effort.) Thus, in SDS terms, Mach's primary target domain is the SDS development and maintenance system; however, it may develop into a candidate for the deployed SDS, too.

In regard to reliability and fault tolerance, the Mach project and two related Mach-based DARPA projects (i.e., Camelot and Avalon) address data integrity, but not processing integrity. Camelot is a distributed transaction facility built on top of Mach. Avalon is built on top of Mach and Camelot; it provides language support.

In regard to security, the Mach kernel utilizes capability-like ports for naming and (discretionary) protection. A related DARPA effort, Trusted Mach, is aimed at achieving multilevel security (in particular, the Orange Book B3 level of protection). Another effort, Strongbox, is exploring the concept of "self-securing" programs, which can run securely on distributed operating systems that provide only minimal security features. Strongbox is built on top of Mach and the transaction facility Camelot.

Status

Mach has a solid technical foundation, due in part to its evolution from RIG[12] and Accent [Rashid 87], earlier projects of Rick Rashid, its principal investigator. Moreover, it has achieved a broad base of interest and support, due not only to its technical foundation, but also to its UNIX compatibility and strong backing from DARPA. Mach is serving as a platform for several interesting distributed system research efforts, many of which are aimed at the SDS requirements of real-time computing, reliability/fault tolerance, and security. These include Real-Time Mach, Camelot, Avalon, Strongbox, and Trusted Mach.[13]

[12] RIG is an acronym for Rochester's Intelligent Gateway (and the preferred designation of the project).

[13] The reader is referred to the Appendix for descriptions of these projects.

Because of its solid technical foundation and broad base of interest and support, Mach represents a valuable resource to the SDS, especially as an operating system for the SDS development and maintenance environment, and possibly as an operating system for ground-based components of the deployed SDS.

Concerns

On the negative side, Mach has not yet achieved independence from UNIX. Mach was produced by modifying and enhancing UNIX, with ideas and experience gained from the RIG and Accent efforts. (Its critics claim that it simply "brings UNIX into the twentieth century," for example, through its advanced virtual memory concepts.) Although a kernelized version of Mach, in which UNIX functionality and code (specifically, the file system and input/output) is removed from the kernel, has been planned for some time, it has not yet been implemented and delivered.

3.4. V

Target Domain

The V kernel was designed for the real-time computing environment. However, it also strives to support interactive computing, as well as large-scale distributed parallel-programmed applications [Cheriton 88b]. Thus, in SDS terms, its primary target domain is the deployed SDS, and its secondary target domain is the SDS development and maintenance system.

In regard to real-time computing, the V project focuses on the traditional aspect, namely, fast response. The V kernel incorporates features typical of current (centralized) real-time operating systems, such as strict priority-based scheduling, accurate time services, and memory-resident programs. In addition, it extends real-time support into the distributed system domain through interprocess communication features such as datagrams, prioritized message transmission and delivery, and conditional message delivery (in which the message is delivered only if the receiver is awaiting a message when the message arrives).

In regard to reliability/fault tolerance and security, the V project adheres to the minimal kernel principle. The V project maintains that simplicity of design is crucial to the development of correct (i.e., trusted) software. The goal is for the kernel to provide the minimal mechanisms necessary for servers to be able to implement their own desired levels of reliability/fault tolerance and security. Some of these mechanisms are incorporated into the Versatile Message Transaction Protocol (VMTP) [Cheriton 88a], the protocol underlying V interprocess communication. Reliability and fault tolerance mechanisms include multicast communication, which enables, for example, communication with multiple providers of the same service or efficient updating of replicates. Security mechanisms include "entity domains"[14] and encryption, which can provide the isolation between security levels required for mandatory protection and the secure authentication of subjects required for discretionary security.

Status

The V project has a solid record of ideas and accomplishments in the form of numerous publications and a mature system. The V kernel is the preeminent minimal kernel. Its only competitor is the kernel of the Amoeba distributed operating system,[15] which is a joint effort between the Centre for Mathematics and Computer Science and the Vrije University, both located in Amsterdam, the Netherlands.

[14] In VMTP, direct communication can occur only on an *intra*-domain basis. The idea is that to implement mandatory security, there would be one domain per security level. Entities can belong to more than one domain, so trusted servers could communicate with users of different domains.

[15] A description of the Amoeba project is provided in the Appendix.

The V kernel is widely recognized in the data communications community, as well as in the distributed operating system community, for its high performance, especially its high-performance interprocess communication [van Renesse 88].

As a mature, high-performance, (traditional) real-time minimal kernel, the V kernel represents a promising base upon which to build a real-time distributed operating system suitable for the deployed SDS.

Concerns

The V distributed system is sometimes viewed as ignoring security and reliability/fault tolerance issues. This perception stems from the V emphasis on minimizing the kernel. However, VMTP has facilities that can support both security and reliability/fault tolerance. The problem is that these approaches to security and reliability/fault tolerance have not been fully developed or implemented. This would involve work above the kernel, assuming that the underlying kernel mechanisms are sufficient. Full development and implementation of the approaches would serve to test (i.e., prove or disprove) the sufficiency of the kernel mechanisms.

4. ONR REAL-TIME COMPUTING INITIATIVE

Since real-time computing is the most pressing SDS distributed operating system research issue, it is important to examine real-time computing as an issue in itself, i.e., to look beyond mainstream distributed operating system research and into the research of the real-time systems community. The point is to identify research relevant to the SDS, as well as to identify means of gaining leverage from it. It is in this vein that the Office of Naval Research (ONR) Real-Time Computing Initiative is addressed here.

In FY 89, the ONR, under the leadership of Andre van Tilborg, is beginning the five-year Foundations of Real-Time Computing Research Initiative [ONR 88]. The objective is to establish a scientific foundation for distributed real-time system development, to remedy the current situation in which *ad hoc* practices prevail. The Initiative is capitalizing on previous ONR-sponsored work. Emphasis is in two areas: (1) specification and verification of real-time systems and (2) real-time scheduling theory. Already, the Initiative is capturing the interests of the real-time systems community and promises to be the focal point of real-time research and development over the next few years.

The Initiative is focusing on the uniprocessor environment initially. Then, in later years, real-time computing in multiprocessor and distributed processing environments will be addressed.

For the sake of contrast to the Alpha approach to real-time computing, let us elaborate upon the approach taken by the Software Engineering Institute and Carnegie Mellon University, in the work that they are doing under ONR sponsorship. The work is based on fixed-priority, rate-monotonic scheduling [Liu and Layland 73], with extensions to address some of problems that are encountered in applying this type of scheduling in practice. Extensions include the following:

- Deferrable server algorithm [Lehoczky 87] [Sha 87], for dealing with aperiodic tasks. Basically, this algorithm preserves some processing bandwidth for aperiodic tasks, while ensuring that periodic tasks meet their deadlines.
- Period transformation method [Sha 87], for dealing with transient overload. The problem is that rate-monotonic scheduling gives the highest priority to the tasks with the shortest periods. In the case of overload, the lowest priority tasks, i.e., the tasks with the longest periods, will miss their deadlines first. But the tasks with the longest periods may actually be the most "important." The period transformation method allows long-period tasks to have artificially high priorities, by having them emulate multiple short-period tasks.

- Priority inheritance protocols [Sha 88], which deal with synchronization and attempt to avoid priority inversion. The basic idea is to have a task that is blocking other tasks execute at the highest priority of the blocked tasks.

5. RECOMMENDATIONS

This section makes recommendations on the directions in which the SDIO should pursue the three distributed operating system projects, Alpha, Cronus, and Mach, that are currently receiving SDIO funding. Then, recognizing that these projects may not meet SDS distributed operating system requirements, especially the real-time requirements, this section goes on to make recommendations aimed at gaining leverage from two other efforts: the V distributed system project and the ONR Real-Time Computing Initiative.

Organized according to project, the recommendations are as follows:

Alpha

The Alpha project is noteworthy for its emphasis on the concept of time-driven resource management. But the noteworthiness stems more from the appeal of the concept than from achievements of the Alpha kernel.

- **Recommendation 1:** Therefore, *the SDIO should concentrate on further investigating the concept of time-driven resource management and not on further developing the Alpha kernel.* Alpha should be viewed only as *a* vehicle (not the only vehicle[16]) for exploring time-driven resource management.
- **Recommendation 2:** The Alpha researchers have assumed that the SDS problem domain is an ideal domain for the application of time-driven resource management. In order to assess the validity of this assumption, *a prototypical SDS problem should be defined and cast in the time-driven resource management framework.* The goals should include the following: (1) examining some of the fundamental assumptions underlying the time-driven resource management approach, such as the assumptions that periodicity and priorities are "artifacts" of out-dated methodologies for real-time system development; (2) measuring the overhead incurred by time-driven resource management; and (3) demonstrating how importance values can be assigned to tasks in a rigorous way.

Cronus

The Cronus project has demonstrated the feasibility of integrating heterogeneous computer systems by imposing a layer of standardization (i.e., the Cronus distributed operating system) on top of native operating systems. It is now time for Cronus to evolve out of the general research domain. The Cronus distributed operating system should be viewed as a "*product*." As a product, it is still subject to enhancement, but it is not deemed to be a promising base for innovative research, especially in the mission-critical, real-time domain. Furthermore, while feasibility of the Cronus approach has indeed been established, utility of the approach has not been. Since the time when the project was initiated, other approaches to integrating heterogeneous systems have been developed.

- **Recommendation 3:** Therefore, further work should be undertaken *only in the context of a specific plan for utilizing Cronus in SDS work.*
- **Recommendation 4:** The National Test Bed (NTB) represents a potential domain for the application of Cronus. RADC and BBN have suggested that the appropriate role for Cronus to play in the SDS would be that of a distributed operating system for development and maintenance activities, such as those of the National Test Bed; and, according to RADC, Martin-Marietta has expressed interest in utilizing Cronus in the NTB. Therefore, *the possibility of utilizing Cronus in the NTB should be pursued*, by encouraging

[16] As noted in Section 3, Mach project plans include further investigation of time-driven resource management.

interaction between the principals, not by continuing general Cronus development.

Mach

Due to its solid technical foundation, its broad base of interest and support, and its strong backing from DARPA, the Mach project represents a valuable resource to the SDIO. It is both a useful platform for research and a promising candidate to serve as a distributed operating system in the SDS development and maintenance system.

- **Recommendation 5:** Therefore, *the SDIO should continue to assist DARPA in supporting the Mach project.* In doing so, it should pursue the actions noted in the following recommendations.
- **Recommendation 6:** In order for Mach to establish its independence from UNIX and to become an even more effective research platform, it is important for the kernelized version of Mach to be completed. *Thus, the completion of the kernelized version should be given high priority.*
- **Recommendation 7:** Since the payoff of having a multilevel-secure distributed operating system, especially for the SDS development and maintenance system, would be high, *the Trusted Mach research effort (on incorporating multilevel security into Mach) should be given high priority.*

V

The V kernel is a mature, high-performance, (traditional) real-time minimal kernel; it represents a promising base upon which to conduct research and to build a real-time distributed operating system suitable for the deployed SDS. Moreover, the V distributed system project has a demonstrated record of research contributions.

- **Recommendation 8:** Therefore, *the V kernel itself, as well as the expertise of its developers, should be taken advantage of, to the extent possible, in the development of the SDS.*
- **Recommendation 9:** *The V kernel should be considered as a distributed operating system base upon which to explore priority-based real-time scheduling policies,* such as those being pursued in the ONR Real-Time Computing Initiative.
- **Recommendation 10:** *The possibility of encouraging the full development of V approaches to security and reliability/fault tolerance should be explored.*

ONR Real-Time Computing Initiative

As the focal point of real-time computing research, the ONR Real-Time Computing Initiative offers promise in overcoming some of the critical real-time issues faced by the SDIO.

- **Recommendation 11:** Therefore, the SDIO should take advantage of the ONR Real-Time Computing Initiative. At the least, *its results should be followed and utilized as appropriate.*
- **Recommendation 12:** *The possibility of the SDIO providing leverage to the ONR research, especially in accelerating research in the directions of multiprocessors and distributed systems, should be investigated.*
- **Recommendation 13:** *The possibility of defining prototypical SDS real-time system problems and offering them as applications to be addressed by ONR research should be investigated.*

References

[Abrams and Podell 87]
Abrams, Marshall D. and Harold J. Podell, *Tutorial: Computer and Network Security*, IEEE Computer Society Press, Washington, D.C., 1987.

[BBN 88a]
BBN Laboratories Incorporated, *Cronus Tutorial Documents, Release 1.2*, January 15,1988.

[BBN 88b]
BBN Laboratories Incorporated, "Release Notice: Cronus Release 1.2," January 15,1988.

[Branstad 88]
Branstad, Martha, Homayoon Tajalli, and Frank L. Mayer, "Security Issues of the Trusted Mach System," *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, December 1988.

[Casey 87]
Casey, Thomas A., Jr., Doug Weber, and Stephen T. Vinter, "The Secure Distributed Operating System Project: Final Report," Report No. 6678, BBN Laboratories Incorporated, October 1987.

[Cheriton 88a]
Cheriton, David R., "VMTP: Versatile Message Transaction Protocol," RFC 1045, SRI Network Information Center, February 1988.

[Cheriton 88b]
Cheriton, David R., "The V Distributed System," *Communications of the ACM 31*, 3 (March 1988), 314-333.

[Dasgupta 88]
Dasgupta, Partha, Richard J. LeBlanc, and William F. Appelbe, "The Clouds Distributed Operating System: Functional Description, Implementation Details and Related Work," *Proceedings of The 8th International Conference on Distributed Computing Systems*, June 1988, 2-9.

[Denning 84]
Denning, Dorothy E., "Cryptographic Checksums for Multilevel Database Security," *Proceedings of the IEEE Symposium on Security and Privacy*, 1984, 52-61.

[Gligor 87]
Gligor, Virgil D., et al., "Traditional Capability-Based Systems: An Analysis of their Ability to Meet the Trusted Computer Security Evaluation Criteria," IDA Paper P-1935, Institute for Defense Analyses, February 1987.

[Graubart 84]
Graubart, Richard D., "The Integrity Lock Approach to Secure Database Management," MTR 9161, The MITRE Corporation, February 1984.

[Gray 79]
Gray, James N., "Notes on Database Operating Systems," *Operating Systems: An Advanced Course*, Springer-Verlag, 1979, 393-481.

[Jensen 85]
Jensen, E. Douglas, Locke, C. Douglass, and Hideyuki Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of IEEE Real-Time*

Systems Symposium, December 1985, 112-122.

[Lehoczky 87]
Lehoczky, John P, Lui Sha, and Jay K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proceedings of 8th IEEE Real-Time Systems Symposium*, December 1987.

[Liskov 88]
Liskov, Barbara, "Distributed Programming in Argus," *Communications of the ACM 31*, 3 (March 1988), 300-312.

[Liu and Layland 73]
Liu, C.L. and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM 20*, 1 (January 1973), 46-61.

[Locke 86]
Locke, C. Douglass, *Best-Effort Decision Making for Real-Time Scheduling*, Ph.D. Dissertation, Carnegie Mellon University, 1986.

[Moss 81]
Moss, J.E.B., *Nested Transactions: An Approach to Reliable Distributed Computing*, Ph.D. Dissertation, Massachusetts Institute of Technology, 1981. (Also MIT Press, Cambridge, Massachusetts, 1985.)

[Nelson and Carroll 87]
Nelson, Victor P. and Bill D. Carroll, editors, *Tutorial: Fault-Tolerant Computing*, IEEE Computer Society Press, Washington, D.C., 1987.

[ONR 88]
Office of Naval Research, Kickoff Workshop, Foundations of Real-Time Computing Research Initiative, November 1988.

[RADC 88]
Rome Air Development Center, Briefing to Phase One Engineering Team (POET) Software Engineering and Operating Systems Panel, 23-24 August 1988.

[Rashid 87]
Rashid, Richard F., "From RIG to Accent to Mach: The Evolution of a Network Operating System," Computer Science Department, Carnegie Mellon University, 28 August 1987.

[Rennels 84]
Rennels, David A., "Fault-Tolerant Computing - Concepts and Examples," *IEEE Transactions on Computers C-33*, 12 (December 1984), 1116-1129.

[Serlin 84]
Serlin, Omri, "Fault-Tolerant Systems in Commercial Applications," *IEEE Computer 17*, 8 (August 1984), 19-30.

[Sha 87]
Sha, Lui, John P. Lehoczky, and Ragunathan Rajkumar, "Task Scheduling In Distributed Real-Time Systems," *Proceedings of IEEE Industrial Electronics Conference*, 1987.

[Sha 88]
Sha, Lui, Ragunathan Rajkumar, and John P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," Departments of CS, ECE, and Statistics, Carnegie Mellon University, 23 May 1988.

[Spector and Swedlow 88]
Spector, Alfred Z. and Kathryn R. Swedlow, editors, *Guide to the Camelot Distributed Transaction Facility: Release 1*, Computer Science Department, Mach/Camelot, Carnegie-Mellon University, Draft of February 4, 1988.

[Stankovic 88]
Stankovic, John A., "Misconceptions About Real-Time Computing: A Serious Problem

for Next-Generation Systems," *IEEE Computer 21*, 10 (October 1988), 10-19.

[SPARTA 88]
SPARTA, Inc, Teledyne Brown Engineering, and The Analytical Sciences Corporation, "Task Order 14 BM/C3 Technology (Multi-Grain) Final Report," 14 October 1988.

[Tanenbaum and van Renesse 86]
Tanenbaum, Andrew S. and Robbert van Renesse, "Distributed Operating Systems," *ACM Computing Surveys 17*, 4 (December 1985), 419-470.

[TCSEC 85]
"Department of Defense Trusted Computer System Evaluation Criteria," National Computer Security Center, DoD 5200.28-STD, December 1985.

[TIS 88]
Trusted Information Systems, Inc., "Trusted Mach Presentation," Ellicott City, Maryland, 7 December 1988.

[TNI 87]
"Trusted Network Interpretation of the TCSEC," National Computer Security Center, NCSC-TG-005, Version-1, July 1987.

[Tokuda 87]
Tokuda, Hideyuki, James W. Wendorf, and Huay-Yong Wang, "Implementation of a Time-Driven Scheduler for Real-Time Operating Systems," *Proceedings of 8th IEEE Real-Time Systems Symposium*, December 1987, 271-279.

[Tripathi 88]
Tripathi, Anand, "An Overview of the Nexus Distributed Operating System Design," Technical Report TR 88-4, Computer Science Department, University of Minnesota, January 1988.

[van Renesse 88]
van Renesse, Robbert, Hans van Staveren, and Andrew S. Tanenbaum, "Performance of the World's Fastest Distributed Operating System," *ACM SIGOPS Operating Systems Review 22*, 4 (October 1988), 25-34.

Appendix

Distributed Operating System Projects

This appendix provides descriptions of the following distributed operating system projects: Alpha, Amoeba, Clouds, Cronus, Heterogeneous Computer Systems (HCS), Mach, and V. The description of each project (1) identifies the goals of the project, (2) gives highlights of the project's technical approach, (3) indicates the current status of the project, and (4) provides a list of references on the project.

It should be noted that the descriptions have been prepared as *summaries* of the referenced literature and, in some cases, of project briefings.[17] As summaries of this material, the descriptions reflect the viewpoints of the project leaders. That is, the descriptions tend to cast the projects in the most favorable terms. Moreover, the descriptions tend to emphasize the same points and features that are emphasized by the project leaders. In short, the descriptions provide (uncritical) overviews of the projects. Critical reviews of the projects are offered in Section 3 of the paper.

[17] During the course of this investigation, the Alpha, Cronus, Mach, and V project leaders gave in-depth briefings on their respective projects to IDA.

Alpha

1. Goals

The Alpha project began in 1985, as part of the Archons Project at Carnegie Mellon University (CMU). Since that time, however, the principal investigator, Doug Jensen, has left CMU and joined Concurrent Computer Corporation. It is anticipated that chief responsibility for the project will follow him from CMU to Concurrent Computer Corporation. The primary sponsor of the Alpha project is the Rome Air Development Center (RADC).

Alpha is an operating system kernel for distributed real-time Battle Management/Command, Control, and Communication (BM/C3) systems. Its goals may be elaborated as follows:

- First, consider the "distributed" aspect. Alpha is aimed at a particular class of distributed systems, namely, *mission-oriented* distributed systems. In a mission-oriented distributed system, system components are physically dispersed, but must be logically integrated into a single system dedicated to the mission. Regarding the degree of physical dispersal, the emphasis in Alpha is on intraplatform, as opposed to interplatform, activities. Alpha's "mission" is the platform's mission.
- Second, consider the "real-time" aspect. Alpha is aimed at a particular class of real-time systems, those dominated by dynamic, stochastic, aperiodic activities with critical time constraints such as deadlines. In the Alpha literature, real-time *BM/C3* systems are identified as being the prototypical systems of this class. Real-time BM/C3 systems may be contrasted to the more traditional real-time systems (represented by low level, closed loop, sampled data applications, such as sensor-actuator feedback control), which are characterized by static, periodic activities.
- Robustness is an inherent requirement of real-time BM/C3 systems, and, as such, is a fundamental goal of Alpha.
- Alpha adopts adaptability as another basic goal. This goal is motivated by the complex and evolutionary nature of BM/C3 systems.

2. Approach

Alpha achieves adaptability through (1) the basic abstractions that define the kernel interface and that extend to the kernel itself, and (2) its strict adherence to the principle of policy/mechanism separation.

The kernel provides mechanisms, but does *not* impose policy. Moreover, the kernel attempts to provide the "right" (i.e., *necessary and sufficient*) mechanisms for the target domain (i.e., distributed real-time BM/C3). These mechanisms include ones (such as atomic transaction support) that are typically left out of "minimal" kernels. The Alpha philosophy is to push more mechanisms down into the kernel to ensure consistency and efficiency and to avoid the costs of recurring implementations.

Basic Abstractions:

- Object: In Alpha, objects are instances of abstract data types. They are passive entities. They are assumed to be of medium to large granularity (i.e., larger than integers). The object abstraction extends to all system services and resources.
- Operation Invocation: Objects are accessed via (and only via) operation invocations. The target of the invocation is specified by a capability.

- Thread: The thread is the active entity in Alpha. It is a logical computation, and the unit of concurrency and scheduling. It corresponds to the *process* in conventional systems. Threads move through objects, independently of the physical locations of the objects, via operation invocations. The thread provides the execution conte t for an operation on an object. Upon operation invocation, the target object is mapped into the context (i.e., virtual address space) of the invoking thread. Thus, the operation is executed according to all the execution attributes (e.g., time requirements, importance, reliability requirements) of the thread.

Interprocess Communication

Alpha uses the Remote Procedure Call (RPC) model of interprocess communication. Operations on objects are invoked via RPCs.

Naming and Protection

Alpha provides object access control, as well as object addressing, through kernel-protected capabilities. In Alpha, a capability consists of a globally unique identifier, a list of operation rights, and a per-operation set of usage restrictions (e.g., no-copy, no-transfer, etc.). No provisions are made for the revocation of capabilities.

Resource Management

Resource management in Alpha is based on three key concepts. First and foremost, Alpha is optimized for the *exceptional* cases rather than the normal (i.e., most frequent) cases. This novel approach to optimization is motivated by the assumption that exceptional cases will *inevitably* occur under stress, which is exactly when it is *most* important for the system to perform its mission. The effect of this optimization approach is that Alpha sometimes pays the price of high overhead in "normal" cases, so that better performance can be achieved in exceptional cases.

Second, Alpha implements *time-driven resource management*. That is, application-specified time constraints are explicitly taken into account in resolving resource contention – for all physical and logical resources (e.g., processors, memory, locks, etc.). In Alpha, time constraints are expressed in terms of "time-value functions," which specify the value to the system of completing an activity as a function of the completion time of the activity. For example, hard deadlines are represented as step functions, whose values go to zero after the deadline.

It should be noted that the first two concepts are closely coupled. Specifically, time-driven resource management is the most important means by which Alpha optimizes for exceptional cases. Alpha's time-driven resource management anticipates that processor (or, more generally, resource) overloads will occur, and ensures that the overloads can be handled gracefully, according to application-specified policies (time-value functions).

Third, Alpha is a *decentralized* operating system kernel, and yet achieves *global* resource management. It does so through the application of the same resource management policies at each node (i.e., through the implementation of the kernel at each node) and through the association of operation invocations with threads. Each operation, whether local or remote, is executed in the context of its invoking thread. Thus, in resolving resource contentions involving the operation, the time constraints and other attributes of the invoking computational activity (i.e., thread) can be taken into account.

Reliability and Fault Tolerance

Alpha provides reliability and fault tolerance through atomic transactions and object replication. In keeping with the principle of policy/mechanism separation, the Alpha kernel implements the three properties of atomic transactions – atomicity, serializability, and permanence – as separable properties. Clients of the kernel can base their policies on various combinations of these properties, rather than being bound to the traditional atomic transaction policy, which bundles all three properties together.

Programming Support

Programming support is minimal. It consists of a preprocessor for the C programming language. The preprocessor provides some primitive extensions (in the form of new keywords) to C that support the programming interface (i.e., abstractions) offered by the Alpha kernel.

3. Accomplishments and Status

- The current version (Release 1) of the Alpha kernel is being developed at CMU directly on Sun workstation hardware.
- Alpha Release 0.5 (a pre-release) has been demonstrated by General Dynamics, in cooperation with CMU, on a BM/C3 application, in particular, coastal air defense.
- One of the major thrusts of the Alpha project over the next few years will be to engage in more cooperative relationships with industry, to facilitate technology transition and evaluation.
- Currently, time-driven resource is applied only to processor scheduling.
- Reliability and fault tolerance are the subjects of ongoing Ph.D. thesis research at CMU. Much work remains to be done, especially on replication mechanisms.
- System services (above the kernel level) have not been developed.
- TCSEC security has not been addressed.

4. References

[Jensen 85]

Jensen, E. Douglas, C. Douglass Locke, and Hideyuki Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of IEEE Real-Time Systems Symposium*, December 1985, 112-122.

[Locke 86]

Locke, C. Douglass, *Best-Effort Decision Making for Real-Time Scheduling*, Ph.D. Dissertation, Carnegie Mellon University, 1986.

[Northcutt 87]

Northcutt, J. Duane, *Mechanisms for Reliable Distributed Real-Time Operating Systems: The Alpha Kernel*, Ph.D. Thesis, published as *Perspectives in Computing 16*, W. Rheinboldt and D. Siewiorek, editors, Academic Press, Inc., 1987.

Amoeba

1. Goals

The Amoeba distributed operating system project has been active since 1984. It is jointly directed by Sape Mullender and Andrew Tanenbaum at the Centre for Mathematics and Computer Science and the Vrije University, both in Amsterdam, the Netherlands.

Amoeba is an object-oriented, capability-based, distributed operating system that supports the flexible sharing of computing power in a cluster of heterogeneous processors, such as Motorola 68010, Intel 8086, NS32016, PDP11/44, and VAX 750's, which are connected by a 10 Mbit/sec star-shaped network. Its goals may be enumerated as follows:

- The Amoeba project adheres to the minimal kernel philosophy.
- The Amoeba project strives for high performance.
- A secondary goal of the Amoeba project is to provide a UNIX-compatible programming environment to its users.

2. Amoeba Approach

The basic computation model adopted in Amoeba is based on object-oriented design concepts. An object is an instance of some abstract data type with some well-defined interface operations that are invoked using the remote procedure call (RPC) mechanism. Each object is assigned a capability that is used for locating the object in the network and also for protection. The remote procedure calls on objects are supported by the Amoeba kernel. Some of the typical system supported objects in Amoeba are directories, files, disk blocks, processes, bank accounts, devices etc.

Architecture

The Amoeba architecture consists of four major components:

- User workstations: One such workstation exists per user. These are used by the users for performing editing tasks and other tasks requiring interactive response.
- Pool Processors: This is a pool of processors that can be dynamically allocated to different tasks as needed. A task is allocated one or more free processors from this pool if it has some CPU-intensive job; these processors are returned to the pool once that activity is complete.
- Server Processors: These are dedicated processors that support the traditional operating system services in the network. These include services for booting, file system, directories, secondary storage blocks, databases, etc. The server processors perform these specialized functions.
- Gateway: All the three components described above are connected by a local area network. Gateways are used to interconnect different, geographically distributed, local area networks.

Each processor in Amoeba runs the same kernel, which basically supports message communication and some other basic services. The approach here was to keep the kernel minimal and to move most of the operating system functions to the application level. One advantage seen in this approach was the flexibility for experimentation with different system primitives at that level.

Interprocess Communication

Interprocess communication in Amoeba is primarily based on the client-server paradigm of computing. To obtain certain service from a server object, a client invokes one of its interface operations using the remote procedure call mechanism. The client is blocked until a response message is received or the timeout period expires. Up to 32K bytes of message can be transferred in the RPC communication.

The message communication supported by the kernel is reliable in the sense that it involves packetization and assembling of messages, detection of lost or duplicate packets, retransmission packets, and processing of acknowledgements. Messages are unbuffered; if a message arrives and no process is waiting for it, then the message is discarded.

In addition to the blocking communication, based on the remote procedure call model, provisions are made for handling emergency messages. This allows the processing of a request at a server to be forced to terminate before its normal completion.

Naming and Protection

Naming and protection in Amoeba is based on the use of sparse capabilities. Each objects is assigned a capability which is 128 bits long. Objects are accessed in the network using this capability; the location of an object is transparent to its clients.

A capability consists of four fields:

- 48-bits service port address of the server managing the object,
- 24-bits object-id assigned to the object by its server,
- 8-bits access rights vector indicating which operations a holder of this capability is permitted to invoke on the object, and
- 48-bits random number that is used for encryption.

The access rights field and the random number field together are encrypted using the 48-bit random number field as the encryption key. Such an encrypted capability is then given to the clients. The server managing the object stores the random number field in its local object table. When a client's request is to be processed, it decrypts the encrypted part of the capability that is presented by a client with its request. If the decrypted random number field matches the number stored in the table, the capability is considered valid. Then only the access rights field is checked to see if the requested operation is permitted.

The scheme used in Amoeba has the advantage that the access through a capability can be revoked by simply changing the random number field in the table. That would make the capabilities based on the old number invalid.

Capabilities for objects, along with the ASCII string names for objects, are stored and managed by the directory servers in the system.

Resource Management

Resource management in Amoeba is done at several different levels: management of activities on a processor, management of pool of processors, and management of servers.

Each Amoeba machine runs a *resource management* process that controls that machine. For the efficiency reasons it executes as a part of the kernel. This process supports functions to read or write segments in the memory of its machine and to create processes. The resource management process also facilitates sharing of code and data segments among multiple concurrent processes on its machine. Thus it is possible to implement a file server using say multiple processes that share disk cache and program code to perform I/O operations. When a request for some file operation is received by the file server, one of such processes, which is free, is assigned to service the request.

The pool of processors is managed by the *process server*, which maintains the allocation status of the processors in the pool. Allocation of free processors is made by the process server. If a particular machine in the pool is multiprogrammed, it manages each virtual processor at that machine as a separate entity.

Reliability and Fault Tolerance

Apart from supporting reliable message communication at the kernel level, the integration of reliability mechanisms in the Amoeba design is minimal. The view taken there is that in most cases users are not willing to pay the high overheads involved in making the system highly fault-tolerant.

A *boot service* is included in Amoeba with which servers in the system can register. This boot service periodically polls each currently active registered server. If no response is received from some server, it assumes it to be dead and requests the process server to create a new copy of it to run on one of the available pool processors.

Typically, a client would retry a request if the timeout condition arises. Meanwhile, if a new server is up, the client's kernel would know this and direct the request to the new server. This scheme provides only minimal amount of fault tolerance and would not function correctly under many different failure conditions.

File System

Access to files is provided by a server that runs above the Amoeba kernel. It supports file system functions that are functionally equivalent to the UNIX system call interface. Another file service supported in Amoeba is called FUSS (Free University Storage System) that allows maintaining multiple versions for files and uses an optimistic concurrency control scheme. This facility is used for implementing updates to file as indivisible atomic actions.

Programming Environment

During the course of this project a strong need was felt to provide a UNIX-compatible programming environment in Amoeba. To achieve this, two server processes are provided: a UNIX file server and a UNIX process server.

As mentioned above, the UNIX file server along with an associated library provides an interface that is very similar to the UNIX (V7) file system. Programs using this library can create and open files, and perform read, write and seek operations. Directory operations such as linking and unlinking files, and mounting and unmounting devices are also provided. Many UNIX programs can be re-linked to the special library and run on Amoeba without any modifications.

For a user running on an Amoeba machine and requiring access to a file on some UNIX machine, a special server is run on the UNIX machines. This server looks like an Amoeba server to the user; it makes capabilities for UNIX files and makes these files accessible to the Amoeba users possessing such a capability.

The UNIX process server supports standard UNIX functions such as *fork*, *exec*, *wait*, *signal*, *kill*, and *exit*. Using these two servers many of the standard programs in UNIX have been re-linked to run on Amoeba. This provides Amoeba users with UNIX environment of *shell*, various editors, C compiler, and some small number of utilities such as *cat*, *grep*, and *sort*.

3. Accomplishments and Status

- The Amoeba system currently includes five different types of CPUs: Motorola 68010, NS32016, Intel 8088, VAX and PDP-11. Almost all of the system services have been implemented and tested. A substantial number of UNIX utilities have been made available in the Amoeba environment. Also, it has been made possible to access UNIX files from Amoeba machines.
- In addition to the system level support to make the system UNIX-compatible, a number of parallel algorithms have been written and tested on Amoeba. These include parallel traveling salesman and parallel alpha-beta search. Also, support for parallel and

distributed compilations has been developed on Amoeba.

- Current research on this project is related to connecting Amoeba machines belonging to different clusters using long-haul networks.

4. References

[Mullender 87]
Mullender, Sape J., ed., *The Amoeba distributed operating system: Selected papers 1984-1987*, CWI Tract 41, Amsterdam, Netherlands: Centre for Mathematics and Computer Science, 1987.

[Mullender and Tanenbaum 85]
"A Distributed File Service Based on Optimistic Concurrency Control," *Proceedings of the 10th Symposium on Operating Systems Principles*, December 1985, 51-62.

[Mullender and Tanenbaum 86]
Mullender, Sape J. and Andrew S. Tanenbaum, "The Design of a Capability-Based Distributed Operating System," *The Computer Journal 29*, 4 (March 1986), 289-300.

[Mullender and van Renesse 84]
Mullender, Sape J. and Robbert van Renesse, "A Secure High-Speed Transaction Protocol," *Proceedings of the Cambridge EUUG Conference*, September 1984.

[Mullender and Vitanyi 85]
Mullender, Sape J. and Paul M.B. Vitanyi, "Distributed Match-Making for Processes in Computer Networks," *Proceedings 4th ACM Principles of Distributed Computing*, August 1985.

[Tanenbaum 84]
Tanenbaum, Andrew S., Robbert van Renesse, and Sape J. Mullender, "Capability-Based Protection in Distributed Operating Systems," *Proceedings of Symposium Certificering van Software*, Ultrecht, Netherlands, November 1984.

[Tanenbaum 86}
Tanenbaum, Andrew S., Sape J. Mullender, and Robbert van Renesse, "Using Sparse Capabilities in a Distributed Operating System," *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986, 558-563.

[Tanenbaum and van Renesse 85]
Tanenbaum, Andrew S. and Robbert van Renesse, "Distributed Operating Systems," *ACM Computing Surveys 17*, 4 (December 1985), 419-470.

[Tanenbaum and van Renesse 87]
"Reliability Issues in Distributed Operating Systems," *Proceedings of the 6th Symposium on the Reliability of Distributed Software and Database Systems*, March 1987, 3-11.

Clouds

1. Goals

The Clouds project was initiated at the Georgia Institute of Technology in 1979. Since then, it has received major funding from NSF, NASA, and RADC. Currently, the Clouds project exists as part of a larger NSF-sponsored project called DARE (for Distributed Application Research Environment).

Clouds is a distributed operating system for a cluster of general purpose computers interconnected by a medium to high speed local area network. Its goals may be elaborated as follows:

- Reliability and fault tolerance: In the beginning, the primary design goal of the Clouds distributed operating system was the support of reliable, fault-tolerant distributed computing. While reliability/fault tolerance remains as a major goal, the emphasis has shifted, as explained in the next paragraph.
- Object/thread model: The object/thread programming model was originally conceived as a means to an end, the end being reliable and fault tolerant distributed computing. However, it has become an end in itself; the support and exploitation of this advanced programming paradigm is now the overriding theme of the Clouds research. Research topics include operating system support for objects, replication and consistency management using objects in a distributed environment, and programming language/methodology/tools support for programming distributed applications using objects.

2. Approach

The Clouds distributed operating system is undergoing a major redesign, from version 1 (v.1) to version 2 (v.2). Clouds v.1 had a monolithic kernel, whereas Clouds v.2 is being designed according to the minimal kernel philosophy. In Clouds v.2, the object/thread model remains as the programming paradigm, and the support of reliable, fault-tolerant distributed computing remains as a major goal. However, the approach to reliability and fault tolerance is undergoing a major evolution.

Architecture

In keeping with the minimal kernel philosophy, the Clouds v.2 distributed operating system consists of the following components:

- Clouds kernel: The Clouds v.2 kernel is referred to as "Ra." The Ra kernel provides the management of segments and virtual spaces needed for persistent objects, along with the location-independent demand paging needed for executing object operations.
- Clouds system services: Clouds system services provide support such as user object management, naming, synchronization, and atomicity. They also provide conventional operating system functions such as device drivers, buffer management, communication protocols, and all I/O related services. These services will be implemented above the Ra kernel by system-provided objects.

Basic Abstractions

Clouds adopts the object/thread model. The object serves as an abstraction of storage, and the thread as an abstraction of computation. Object invocations serve as the integrating mechanism. These abstractions are summarized below:

- Object: In Clouds, an object is an instance of an abstract data type. It is a passive entity, in particular, a persistent virtual address space. It is used to encapsulate all data, programs, devices, and resources.
- Object Invocation: Objects are accessed via (and only via) object invocations, to operations defined on the objects.
- Thread: The thread is the active entity in Clouds, the unit of computation and concurrency that is used to execute the code in objects. Threads traverse objects, independently of machine boundaries, via object invocations. Threads are implemented as lightweight processes. A thread that spans machine boundaries is implemented by several processes, one per machine.

Interprocess Communication

Clouds provides two modes of interprocess communication, both based on object invocation. In particular, objects can be invoked using either one of two mechanisms: remote procedure call (RPC) or distributed shared memory (DSM). Using RPC, the thread migrates to the home site of the object and executes there; using DSM, the invoked object is demand paged to the site of the invoking thread. The mechanisms have orthogonal advantages and can be chosen for optimum performance.

Naming and Protection

Clouds utilizes capabilities for object naming. Each Clouds object is named and accessed by its capability, which is globally unique and location-independent.

At this point in time, protection is not a goal of the Clouds project. Therefore, although capabilities could be utilized for protection as well as for naming, they currently are not being utilized for this purpose.

Storage Management

In Clouds, emphasis is placed on the object as an abstraction of storage. The object is viewed as unifying the concepts of file space (long-lived storage) and memory space (volatile storage, but essential for computation), by providing a *persistent* virtual address space. Since objects provide permanent storage, the need for a traditional file system is eliminated; the file system is replaced by *object memory*. Object memory is stored on disk and demand paged. The demand paging happens with storage on the local machine, if the invocation uses RPC. The demand paging occurs over the network if the invocation uses DSM.

Resource Management

The Ra kernel manages the low-level scheduling of threads, demand paging, and segment and memory allocation. All other resource management tasks are done at the higher level through system objects. Currently, the system objects under implementation will do object management, task management, naming, and partition management. More will be implemented as the system evolves. One of the points of the Ra approach is to be flexible and avoid being locked into any particular resource management scheme.

Reliability and Fault Tolerance

Clouds v.1 supports absolute consistency (i.e., traditional atomic transactions), and allows customized synchronization and recovery for applications that cannot tolerate the performance penalties incurred by absolute consistency semantics. Clouds v.2, on the other hand, is being designed to offer a range of *consistency-preserving* mechanisms, from "best effort" to absolute consistency.

The consistency preserving mechanisms are based on attaching consistency labels to the operations declared in the objects. The labels allow the operations to update the objects with (1) transaction-like semantics, for preserving inter-object consistency of data, (2) locally atomic semantics for preserving the consistency of data locally within one object, or (3) best-effort semantics like the way processes in conventional systems update memory and files.

Programming Support

The Clouds project provides programming support in the form of a programming language referred to as Aeolus. Aeolus is viewed as the first generation language for Clouds. It currently supports the features of Clouds v.1, but is being expanded to support Ra and Clouds v.2. Furthermore, it does not support features such as inheritance and subclassing, although such support is currently under consideration.

Aeolus is most like Modula-2 and Ada in its syntactic style and the kinds of features it supports. A central feature of the language is the ability to declare several kinds of objects, ranging from single-instance library objects (like standard modules and packages in Modula-2 and Ada), through abstract data types with multiple instances managed by the Aeolus runtime support (like clusters in Clu), to persistent Clouds objects with instances managed by the Clouds kernel. This range of alternatives allows a programmer to choose the level of functionality (and associated management overhead) appropriate for each object used in an application. Supporting such a choice is a recurrent theme in the design of Clouds.

User Interfaces

In [GIT 86], the Clouds researchers suggest the need for Clouds-UNIX interoperability, of two distinct flavors. First, Clouds services should be made available to UNIX users and programs, through a Clouds library on UNIX, in a way that would enable a cluster of Clouds machines to serve as a back-end distributed system to UNIX workstations. Second, established UNIX services (e.g., mail, text processing, etc.) should be made available to Clouds users, through a "UNIX gateway."

In [Dasgupta 88], the Clouds researchers suggest an X-windows interface to Clouds as well.

3. Clouds Accomplishments and Status

- Clouds v.1, the monolithic kernel version, became operational in 1987. It is implemented on VAX-11/750s, interconnected by a 10-megabit Ethernet.
- Clouds v.2, the minimal kernel version, is being implemented on SUN 3 workstations, also interconnected by Ethernet. More specifically, the implementation of the Ra kernel is complete, and the implementations of system services are underway.
- The Clouds v.2 Ra kernel is implemented in C++.
- Ongoing Ph.D. Thesis research is addressing the following topics: (1) the Clouds v.2 approach to reliability and fault tolerance; (2) distributed shared memory techniques, including protocols for providing coherence for concurrent DSM invocations, performance studies of DSM vs. RPC, and defining support for faster local and remote object invocation under RPC and DSM; and (3) object location techniques, in particular, support for location independent object invocation in a large network where objects migrate, utilizing multicasting and statistical techniques based on past data.
- Other research is addressing the following topics: (1) better techniques for providing low-level system services in a multithreaded operating system; and (2) fault tolerance using replicated data and computation.

4. References

[Bernabeu Auban]
Bernabeu Auban, Jose M., et al., "The Architecture of *Ra*: A Kernel for *Clouds*," School of Information and Computer Science, Georgia Institute of Technology.

[Dasgupta 88]
Dasgupta, Partha, Richard J. LeBlanc, and William F. Appelbe, "The Clouds Distributed Operating System: Functional Description, Implementation Details and Related Work," *Proceedings of The 8th International Conference on Distributed Computing Systems*, June 1988, 2-9.

[GIT 86]
The School of Information and Computer Science, Georgia Institute of Technology, "Effective Distributed Computing: A Reliable Object-Based Environment for Computer Science Research," A Proposal to the National Science Foundation's Co-ordinated Experimental Research Program, September 15, 1986.

[Pitts and Dasgupta 88]
Pitts, David V. and Partha Dasgupta, "Object Memory and Storage Management in the Clouds Kernel," *Proceedings of The 8th International Conference on Distributed Computing Systems*, June 1988, 10-17.

Cronus

1. Goals

Cronus has been under development at BBN Laboratories since 1981. It is sponsored by the Rome Air Development Center (RADC),

Cronus is a distributed operating system for interconnecting heterogeneous computer systems. Typically, the computer systems fall under a common administrative domain, and are interconnected by one or more high-speed local area networks. The computer systems may also be interconnected by wide area networks, via an internet (such as the DARPA Internet). Each set of computer systems is called a "cluster." The initial focus of Cronus has been on intracluster communication and cooperation; however, more recently, consideration has been given to intercluster aspects. The goals of Cronus may be elaborated as follows:

- The ultimate goal of Cronus is to integrate *heterogeneous* computer systems into an effective general-purpose distributed computing environment for the development and execution of large-scale applications.
- Heterogeneity is the key concept. The hallmark of Cronus is its support of heterogeneity - of both hardware and software resources. The motivation for this emphasis is threefold: (1) to allow applications and users to take advantage of the unique functionality offered by various hardware and software resources, (2) to allow existing software to continue to be used, and (3) to allow familiar computing environments to continue to be used.
- In particular, Cronus is designed to interoperate with, rather than to replace or totally encapsulate, constituent (i.e, native) operating systems.
- In addition to heterogeneity, the Cronus project places major emphasis on providing comprehensive support for large-scale distributed application development.

2. Approach

The Cronus approach is to introduce layers of software - the Cronus distributed operating system - on top of constituent operating systems (or, in some cases, on bare hardware). The Cronus distributed operating system is based on the object model; each system resource is a typed object, and is accessed through operations defined by the type. The object model provides an extensible architecture, in that application developers can cast application-specific resources in terms of new object types, which can be defined as subtypes of existing types.

The Cronus distributed operating system supports heterogeneity by serving as a bypassable layer of abstraction between application programs and constituent operating systems. Through this approach, application programs gain access to a coherent, uniform (object-oriented) system interface, regardless of computer system base; however, they also retain conventional access to constituent operating system resources and services.

Architecture

The Cronus distributed operating system consists of the following components:

- Cronus kernel: The Cronus kernel supports the Cronus object model. Namely, it implements the basic abstractions of object, operation invocation, and (Cronus) process, as defined below. It must be installed and run on each host participating in the Cronus distributed system. Typically, it is implemented as an application process of the constituent operating system.

- Cronus system services: Cronus system services provide the traditional operating system services, plus additional services specifically designed for the support of distributed application development. Each system service is implemented by one or more manager processes (i.e., servers), which run above the Cronus kernel as Cronus processes. Current system services include an authentication service, a catalog service, a configuration service, a file service, and a type definition service.

The distributed computing architecture supported by Cronus includes the following components as well:

- Application services: An application service is one or more processes developed by application programmers to manage the resources that make up applications. An application is typically composed of several services responsible for several different object types.
- Clients: Clients are processes that use services. While any service may act as a client to another service, most clients are processes that interact directly with users, such as user commands, utilities, and application-specific graphical user interfaces.

Basic Abstractions

Since Cronus is based on the object model, the basic abstractions are *objects* and *operation invocations*. To implement the object model, the Crouus kernel introduces the *process* as a kernel-supported object type. Thus, the basic abstractions of Cronus are the following:

- Object: In Cronus, an object is an instance of an abstract data type, where a type can be defined as a subtype of a parent type, and hierarchical inheritance is supported. Objects are passive entities.
- Operation Invocation: Objects are accessed via (and only via) operation invocations. (This abstraction is inherent in the object abstraction.)
- Process: Processes are the active entities in Cronus. They are used to implement object managers, as well as application programs that execute on Cronus. An object manager is the entity that is responsible for manipulating all of the objects of one or more given types on a given host using the operations defined by the types. The Cronus system managers are simply Cronus-provided object managers, for Cronus-defined object types. The Cronus process abstraction corresponds to the process abstraction found in conventional operating systems, and is typically implemented as a constituent operating system process that executes in user space.

Interprocess Communication

Cronus interprocess communication (IPC) is designed to support operation invocations from clients to object managers, where the invocations can be synchronous or asynchronous, and can have one or many targets. It is implemented as a series of layers.

At the lowest layers, collectively referred to as the *network* layer, are standard data communication protocols, which are typically implemented by the constituent operating systems. Currently, Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Protocol (IP), and Ethernet are utilized. However, other protocols could be substituted easily.

Above the network layer is the layer designated as the *IPC* layer. This layer implements three communication primitives: *Invoke, Send,* and *Receive.* In a typical scenario, Invoke would be used by a client process to invoke an operation on an object. Using Invoke, the client references the object by name (not location, thereby ensuring host-independent, network-transparent access to objects), and this causes a message to be sent to the process serving as object manager of the target object. The object manager would retrieve the message from its message queue via the Receive primitive, perform the requested operation, and then send a reply to the client via the Send primitive. The operation would actually be performed by a *lightweight* process (or task, in Cronus terminology) created by the object manager; thus, operations can be performed concurrently. Finally, the client would receive the reply via the Receive primitive. The separation of the client's Invoke from the subsequent Receive allows for asynchrony and concurrency. It should be noted that the Send is simply an optimization of the Invoke. It allows a message to be sent directly to a process, instead of to the process manager.

Above the IPC layer is a layer designated as the *message encodement* layer. This layer is responsible for encoding and decoding messages, using canonical (system-independent) data representations. Cronus defines canonical data representations for many common data types and structures. It also offers extensibility by supporting the creation of new canonical types from existing ones.

At the highest layer is a protocol designated as the Operation Protocol. This layer defines a set of standards for interpreting messages between clients and managers, and supports a synchronous remote-procedure-call-like (RPC-like) programming interface for operation invocation.

Naming

Cronus has a two-level naming system. At the high level is a hierarchical symbolic name space. At the low level is the flat name space of Unique Identifiers (UIDs). A UID is a 96-bit object identifier, which is guaranteed to be unique over all objects over all time within a cluster; sixteen bits of the UID specify the object's type, and the remaining bits establish uniqueness. The Cronus catalog, which is implemented as a distributed entity by the catalog managers, provides the mapping between symbolic names and UIDs.

Protection

In Cronus, protection is achieved through access control lists. The access control list for an object specifies which users or *groups* of users have which access rights to the object. Privileges associated with access control lists can be defined separately for each object type. These privileges are specified by the application developer, allowing access controls to be customized for each type. Authentication (of the identity of a user) is implemented by an authentication manager, which subjects a user to a password-based authentication procedure upon login.

Resource Management

In Cronus, global resource management is approached according to the principle of policy/mechanism separation. That is, Cronus provides mechanisms, and the mechanisms enable object managers to cooperatively enforce object-type-specific policies. The mechanisms include: (1) the ability of object managers to query the status of their *peer* object managers, one of which must be installed at each host where objects of the given type exist, (2) the ability of object managers to redirect requests to peer object managers, and (3) the ability of applications to indicate preferred hosts. These mechanisms support high-level resource management; low-level resource management is performed by the constituent operating systems. These mechanisms have been used in several services to implement specific management policies, such as dynamic load balancing during Cronus file creation.

Reliability and Fault Tolerance

Cronus supports object migration and object replication. With respect to replication, the Cronus project has recently adopted the philosophy of application-specific replication management. Namely, Cronus has progressed from an inflexible weakly consistent replication strategy to a flexible "version voting" replication strategy. In the weakly consistent replication strategy, updates were propagated on a best-efforts basis, and object managers would periodically (e.g., upon their host coming back up after being down) utilize Cronus-provided mechanisms to bring their copies up to date. In the version voting replication strategy, version vectors (one for each replicated object, giving host location and version number pairs) are used to keep track of copies and consistency, and read and write quorums can be set to provide the application-desired balance between availability and consistency.

Cronus delivers replication support to application developers through its object manager programming support tools. When specifying a new object type, the application developer defines a replication policy by selecting a Cronus-supported replication strategy and then specifying values for the parameters of the selected strategy. Currently, only one replication strategy, version voting, is available, but it is envisioned that more will be developed, as demanded by applications. Based on the object type definition, Cronus automatically generates the code that implements the specified replication policy.

Cronus can also dynamically locate objects on invocation, ensuring that clients will always be able to access a copy of an object (providing one is available).

Atomic transaction support is being investigated in the context of distributed database management systems, as a part of the Cronus Distributed Database Management System Project.

Programming Support

The fundamental assumption underlying Cronus programming support is that large-scale applications will be developed in accordance with the object model, just as the Cronus distributed operating system itself is. Under this assumption, the key to application development is the definition of new object types to represent application-specific resources and the development of new object managers to embody the newly defined object types. Therefore, Cronus programming support focuses on automating the process of developing new object managers. In particular, Cronus seeks to relieve the application developer's coding burden through the use of a non-procedural program development specification language. Cronus takes non-procedural specifications of a new object type, and automatically generates code for skeletal object managers (including multitasking for concurrent operation processing, message parsing and validation, access control checks, operation dispatching, data conversion between canonical and system-specific data representations, and stable storage management), as well as for RPC client stubs. The code generation process relies upon the Cronus libraries; the skeletal object managers incorporate procedure calls to Cronus library routines for many functions (e.g., data conversion between the canonical and system-specific representations of common data types). The application developer completes the object manager by providing routines that implement the operations defined by the new object type.

Cronus programming support also includes (1) extensive subroutine libraries, including interprocess communication routines, data conversion routines, and RPC interfaces to Cronus objects; (2) a set of user commands; (3) a set of operator commands; (4) operations inherited by all objects, for access control, monitoring and control, debugging, and replication and migration support; (5) a program to be used in conjunction with a local debugger, to assist in object manager debugging; (6) source management control software; and (7) a bug tracking facility.

3. Accomplishments and Status

- The original Cronus cluster has been operational, at BBN, since 1984. Recently, additional clusters have been established at RADC, MITRE Bedford, Carnegie-Mellon University, Honeywell Minneapolis, BBN San Diego, and NOSC. Intercluster communication is supported via the DARPA Internet.
- Cronus has achieved high portability. It is written in the C programming language. Machine-dependent code is confined to a few modules. Cronus has been ported to a new machine in as little as two man-weeks.
- Programming support was initially focused on C, but it is now being extended to Common Lisp and Ada. Application components have also been written in FORTRAN.
- Cronus implementations exist for the following systems: DEC VAX with VMS, Ultrix, and BSD Unix; SUN 2 and Sun 3 with Sun UNIX; MASSCOMP with RT UNIX;

Symbolics Lisp Machine with Genera; and IBM PC/AT with SCO Xenix. Cronus implementations are planned for multiprocessor architectures.

- Distributed applications that have been developed on Cronus include a navy target tracking application, a distributed simulation of SDI boost phase, and numerous office automation applications.
- Multilevel security was investigated in a research project, the Secure Distributed Operating System (SDOS) Project. Among the conclusions of the project was the following [Casey 87, p.19]: "Thus, the host operating system(s) on top of which SDOS [i.e., *secure* Cronus] is implemented must have a minimum of a B2 rating, and ratings of B3 or A1 are more desirable." GEMSOS, a product of Gemini Computers, Inc., of Carmel, California, was selected as the best candidate for serving as a multilevel secure constituent operating system.
- Distributed databases are being addressed in an on-going research project, the Cronus Distributed DBMS Project.
- The Cronus developers and sponsors have suggested that the appropriate role for Cronus to play in the SDS would be that of a distributed operating system for development and maintenance activities, such as those of the National Test Bed.

4. References

[BBN 88a]
BBN Laboratories Incorporated, *Cronus Operator's Reference Manual, Release 1.2*, January 15, 1988.

[BBN 88b]
BBN Laboratories Incorporated, *Cronus Programmer's Reference Manual, Release 1.2*, January 15, 1988.

[BBN 88c]
BBN Laboratories Incorporated, *Cronus Tutorial Documents, Release 1.2*, January 15,1988.

[BBN 88d]
BBN Laboratories Incorporated, *Cronus User's Reference Manual, Release 1.2*, January 15, 1988.

[BBN 88e]
BBN Laboratories Incorporated, "Release Notice: Cronus Release 1.2," January 15,1988.

[Berets 85]
Berets, James C., Ronald A. Mucci, and Richard E. Schantz, "Cronus: A Testbed for Developing Distributed Systems," *IEEE Military Communications Conference*, October 1985, 409-417.

[Berets and Sands 87]
Berets, James C. and Richard M. Sands, "Introduction to Cronus: A Distributed Operating System," Draft Paper, BBN Laboratories Incorporated, January 1987.

[Casey 87]
Casey, Thomas A., Jr., Doug Weber, and Stephen T. Vinter, "The Secure Distributed Operating System Project: Final Report," Report No. 6678, BBN Laboratories Incorporated, October 1987.

[Dean 87]
Dean, Michael A., Richard M. Sands, and Richard E. Schantz, "Canonical Data Representation in the Cronus Distributed Operating System," *Proceedings of the IEEE Infocom '87*, March 1987, 814-819.

[Dean 88]
Dean, Mike, "Cronus, A Distributed Operating System: Ada Integration Investigation," Cronus Project Technical Report No. 7, Report No. 6797, BBN Laboratories Incorporated, April 1988.

[Gurwitz 86]
Gurwitz, Robert, Michael A. Dean, and Richard E. Schantz, "Programming Support in the Cronus Distributed Operating System," *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986, 486-493.

[Schantz 85]
Schantz, R., et al., "CRONUS, A Distributed Operating System: Phase 1 Final Report," Report No. 5885, BBN Laboratories Incorporated, January 1985.

[Schantz 86a]
Schantz, R., et al., "CRONUS, A Distributed Operating System: Cronus DOS Implementation, Final Report," Report No. 6183, BBN Laboratories Incorporated, March 1986.

[Schantz 86b]
Schantz, Richard E., Robert H. Thomas, and Girome Bono, "The Architecture of the Cronus Distributed Operating System," *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986, 250-259.

[Vinter 87]
Vinter, Stephen T., et al., "The Cronus Distributed DBMS Project: Functional Description," Report No. 6660, BBN Laboratories Incorporated, October 1987.

Heterogeneous Computer Systems (HCS) Project

1. Goals

The HCS Project was initiated at the University of Washington in 1985. It has received major support from the National Science Foundation.

To avoid misleading the reader, let us begin this discussion by acknowledging that the Heterogeneous Computer Systems (HCS) Project is not building a distributed operating system. Instead, it is investigating an approach to the integration of heterogeneous computer systems that can, to some extent, be viewed as a competitor to the distributed operating system approach (especially as exemplified by Cronus). Stated succinctly, the goal of the HCS project is to *facilitate* the *loose integration* of *heterogeneous computer systems*. This goal may be elaborated by considering its three facets:

- Heterogeneous computer systems: As its name implies, the project is directed toward the aspect of heterogeneity. Heterogeneity – of hardware, as well as software – is viewed as being both inevitable and desirable, especially in environments, such as research labs, where new systems are acquired precisely because of their unique functionality and capabilities.
- Loose integration: The HCS project aims for (in their words) "loose integration," which is meant to offer more distributed computing support than network services such as remote login and file transfer, but less than true distributed operating systems. Resource sharing is supported, but not always made network-transparent.
- Facilitate: The last facet of the goal is that the HCS project is seeking to significantly ease the process of incorporating new systems into an existing HCS environment.

2. Approach

The HCS approach represents a distinct departure from the prevailing wisdom that the way to deal with heterogeneity is to introduce standardization. It is based on emulation and accommodation. Instead of defining new standards that must be implemented by all systems, the HCS project strives to build software that accommodates multiple standards and can emulate a wide range of existing facilities. Below, we present an overview of the HCS architecture, and then illustrate the HCS techniques of emulation and accommodation by considering how the techniques are applied to the implementation of remote procedure call (RPC).

Architecture

The HCS architecture is organized as a lower layer of underlying facilities and an upper layer of fundamental network services. The underlying facilities are RPC and naming. The network services are remote computation, mail, and filing. The highlights of the underlying facilities and network services are concisely captured in the following summaries, which are extracted from [Notkin 88, p. 259]:

- The HCS RPC (HRPC) facility utilizes a modular design that, by appropriate selection of implementations at run time, can be made to emulate a wide variety of existing RPC facilities. Thus, the central core of HCS – those systems on which the HRPC facility has been implemented – can easily be adapted to communicate with a new system type.
- The HCS name service (HNS) creates a global name space that accesses names and data from existing name services. By using data in existing name services, rather than reregistering data into an entirely new name service, existing clients can work with their name

services without change, and new clients of HNS need not make changes when a new underlying name service is introduced.

- The HCS remote computation service provides a generic mechanism by which services can be executed remotely. Each remote service includes a description of its required inputs and outputs, the steps needed to process the information, and the steps required to create an environment in which to execute the service. These descriptions are processed by interpreters that are responsible for passing information between nodes and for performing any necessary translation of file names, options, etc.
- The HCS mail service (HMS) attempts to improve the quality of most existing mail services while integrating services that are based on diverse models. The mail service is structured like the Xerox Grapevine mail service [Birrell 82], but also integrates mail systems such as UNIX's *sendmail* [Allman 83]. Abstract mail retrieval and submission interfaces are defined and implemented in multiple ways, facilitating the integration of new mail systems.
- The HCS filing service is represented by two distinct efforts: The first approach defines a centralized filing service that stores files in multiple representations (based on those used in the HRPC facility). The second approach is based on that of the naming facility, where existing local file systems are used to store data, and neither the files themselves nor information about them (such as the file type) need be reregistered.

Remote Procedure Call

The HCS project *factors* RPC into five components, with clean interfaces among the components. The components are: (1) compile-time support, (2) bind-time protocol (e.g., SUN RPC, Xerox Courier), (3) (call-time) transport protocol (e.g., UDP, TCP, Xerox XNS), (4) (call-time) control protocol (e.g., SUN RPC, Xerox Courier), and (5) (call-time) data representation (e.g., SUN XDR, Xerox Courier).

An HRPC client or server and its associated stub, which are generated at compile time, are designed to accommodate multiple bind-time and call-time protocols. The choice of which bind-time and call-time protocols to actually use in a particular scenario is made at bind time. Thus, an HRPC server can communicate with unmodified native RPC clients through emulation, by choosing the bind-time and call-time protocols utilized by the native RPC of the clients; similarly, an HRPC client can communicate with native RPC servers through emulation.

3. Accomplishments and Status

- As stated at the beginning of this summary, the HCS project is not building a system. Rather, it is developing and demonstrating, as well as utilizing, approaches to the integration of heterogeneous computer systems. To a large extent, it is seeking to introduce order into what has been an inconvenient, expensive, time-consuming, and ad hoc process.
- According to the authors of [Notkin 88], the HCS approaches are proving to be effective in meeting the demands of the University of Washington's Department of Computer Science, which has over fifteen significantly different hardware/software systems.

4. References

[Allman 83]
Allman, E., "Sendmail - An Internetwork Mail Router," *UNIX Programmer's Manual 4.2BSD, 2C*, August 1983.

[Birrell 82]
Birrell, A.D., et al., "Grapevine: An Exercise in Distributed Computing," *Communications of the ACM 25*, 4 (April 1982), 260-274.

[Black 85]
Black, Andrew P., et al., "An Approach to Accommodating Heterogeneity," Tech. Rep. 85-10-04, Department of Computer Science, University of Washington, Seattle, October 1985.

[Notkin 88]
Notkin, David, et al., "Interconnecting Heterogeneous Computer Systems," *Communications of the ACM 31*, 3 (March 1988), 258-273.

Mach

1. Goals

The Mach project was initiated at Carnegie Mellon University (CMU) in 1984 as the operating system effort of DARPA's Strategic Computing Initiative (SCI). Mach was envisioned as an operating system that would (1) provide a uniform (UNIX-compatible) software base across the architectures existing at the time, as well as the new advanced architectures being developed as part of the SCI, and (2) support the interconnection of these architectures into distributed computing environments. Its goals may be elaborated as follows:

- Mach was designed to extend UNIX functionality to multiprocessor architectures, ranging from (1) uniform access, shared memory multiprocessors (UMA, for Uniform Memory Architecture) (e.g., Encore Multimax, Sequent Balance), to (2) differential access, shared memory multiprocessors (NUMA, for non-UMA) (e.g., BBN Butterfly, IBM RP3), to (3) multicomputer architectures (NORMA, for No Remote Memory Access Architecture) (e.g., hypercube).
- Mach was designed to extend UNIX functionality to large memory architectures.
- Mach was designed to extend UNIX functionality to distributed computing environments, in which diverse architectures (i.e., uniprocessors, multiprocessors) interconnected by high speed networks support distributed applications.
- To take advantage of the vast supply of UNIX-based software, Mach was designed to offer (and continues to offer) UNIX compatibility (specifically, binary compatibility with 4.3 BSD).

2. Approach

Although Mach offers UNIX compatibility, it is *not* intended to be bound to UNIX. The current, evolved vision is for the Mach distributed operating system to be based on a *minimal* kernel upon which multiple operating system environments can be built. At this point, the kernelization is not complete, and some UNIX functionality is still embedded in Mach kernel code. When the kernelization is complete, it will be possible to emulate operating system environments other than UNIX 4.3 BSD on top of the Mach kernel.

Architecture

The goal is for the Mach distributed operating system to evolve as two layers:

- A small, extensible kernel layer, namely, the Mach kernel.
- An operating system environment layer built on top of the kernel layer. The operating system environment layer is to be realized by user-state tasks. The user-state tasks that run on top of the Mach kernel could be designed to emulate UNIX, as well as other established and/or interesting operating system environments.

Basic Abstractions

The Mach kernel is based upon five inter-related abstractions:

- Task: unit of resource allocation – includes a virtual address space and a set of port rights (capabilities).
- Thread: unit of computation – a lightweight process – maintains processor state (e.g., hardware registers) necessary for independent execution. It should be noted that a

UNIX process corresponds to a task with a single thread of control.

- Port: a simplex communication channel, implemented as a kernel-protected message queue.
- Message: typed collection of data objects.
- Memory object: secondary storage object that is mapped into a task's virtual memory.

Interprocess Communication

Mach interprocess communication (IPC) is based on the port and message abstractions. Ports are the *reference objects* in Mach, and, as such, are viewed as playing the same role as capabilities in an object-oriented system. Objects such as tasks, threads, and memory objects are represented as ports, and operations on these objects are performed by sending messages to the ports that represent them. Only tasks with *send rights* to a port can send messages to it, and only the (single) task with *receive rights* to a port can receive messages from it.

Messages can be sent and received synchronously (as in Remote Procedure Calls (RPCs)) or asynchronously. They can contain capabilities. In fact, the only way for a task to acquire a capability is to receive it in a message.

In Mach, the kernel itself implements *local* IPC only. However, a user-state task, called the *network message server*, transparently extends IPC into a network environment. This task maintains mappings of local "proxy" ports to global "network" ports. It forwards messages using network protocols of its choice.

Naming and Protection

As noted in the IPC section, the Mach kernel uses capabilities, in the form of ports, for naming and protection on a single system.

The network message servers extend the protection to the network environment, by implementing mechanisms to protect both the messages sent over the network to network ports and the network port capabilities.

Security – Trusted Mach

The Trusted Mach project is a DARPA-sponsored research effort of Trusted Information Systems, Inc. The goal is to build a version of Mach – Trusted Mach – that meets the B3 level of protection as specified in the National Computer Security Center (NCSC) Trusted Computer System Evaluation Criteria (TCSEC), the so-called "Orange Book" [TCSEC 85].

The project adopts the idea of "incremental reference monitors." At the lowest level is the Trusted Mach Kernel. At the intermediate level is the reference monitor composed of the kernel and a trusted name server. At the highest level is the reference monitor composed of the kernel, a trusted name server, and other trusted servers. Thus far, work has concentrated on the kernel level of a single machine. Mach's ports are serving as the protected objects in Trusted Mach; its tasks (through their threads, which are the active entities) are serving as the subjects. Extensions are being developed to meet the TCSEC requirements for both discretionary and mandatory protection.

At this time, the Trusted Mach project is utilizing a Spring 1988 version of Mach. Since this version is not kernelized, the effort cannot yield a trusted operating system. The unkernelized version of Mach is serving as a platform for research into multilevel security, not as a base upon which to build a trusted system. The development of a trusted version is tied to the completion of Mach kernelization.

Security – Strongbox

Strongbox is built on top of Camelot and Mach. It is based upon the new concept of "self-securing" programs, i.e., programs that can run securely on distributed operating systems (such as Mach) that provide only minimal security facilities.

Two key algorithms implemented by Strongbox are zero knowledge authentication and fingerprinting.

It should be noted that Strongbox is (currently) concerned with the security issues that arise from protecting the privacy of data and ensuring the integrity of data from alteration; security issues of denial of service, covert channel analysis, and traffic analysis of message patterns have not been considered, although they could be.

Storage Management

Mach places major emphasis on virtual memory management, especially in the areas of portability, advanced functionality, and memory/communication integration. In regard to portability, Mach virtual memory management assumes minimal hardware support, and is carefully constructed to isolate machine-dependent code into a single module. Notably, it achieves improved performance, even while it minimizes hardware dependencies.

In regard to advanced functionality, Mach supports large, sparse virtual address spaces; memory mapped files; shared libraries; copy-on-write virtual copy operations; copy-on-write and read/write memory sharing between tasks, through inheritance (which is specified on a per-page basis as shared, copy, or none) of memory regions from a parent task to a child task; and user-provided memory objects and pagers.

In regard to memory/communication integration, the Mach project emphasizes the complementary roles that memory and communication can play. Namely, Mach uses memory mapping techniques (i.e., copy-on-write sharing) to accomplish communication; an entire address space may be sent in a single message with no actual data copy operations performed. In the other direction, Mach implements virtual memory through its IPC facilities; in particular, it maps process addresses onto memory objects, which are represented by ports and accessed via messages. This is what enables user-provided memory objects.

Resource Management – Real-Time Mach

Real-Time Mach provides an integrated time-driven scheduler, with support for both periodic and aperiodic threads. Rate monotonic scheduling policies are used for periodic threads. Value function scheduling policies (derived from Locke's thesis, as was Alpha's) are used for aperiodic threads. Real-Time Mach uses piecewise linear approximations to continuous value functions for efficiency. For a collection of periodic and aperiodic threads, the periodic threads are scheduled first, and then the aperiodic on a best effort basis.

Real-Time Mach implements policy/mechanism separation. Currently, seven scheduling policies are implemented. Different applications or experiments can utilize different policies.

Tools and a test bed have been developed to support Real-Time Mach. They allow workloads to be specified, and schedules to be constructed, examined, simulated, and monitored.

Currently, Real-Time Mach has been applied only in a uniprocessor environment and only to CPU scheduling. Plans call for it to be applied in a multiprocessor environment and to other resource types (e.g., memory, I/O). Also, impacts of interactions (requiring synchronization) among threads remain to be considered.

Reliability and Fault Tolerance – Camelot and Avalon

Camelot is a distributed transaction processing facility built on top of Mach. As such, it addresses the requirements of reliability and fault-tolerance. Its basic abstraction is the transaction. A transaction is a collection of operations that exhibits three properties: atomicity, permanence, and serializability.

Avalon is built on top of Camelot and Mach. It is implemented as a preprocessor for C++. It provides language support for reliable distributed systems based on atomic transactions.

Programming Support

An interface specificaticn language, MIG (Mach Interface Generator), has been developed for Mach. MIG generates C or Common Lisp RPC stubs.

3. Accomplishments and Status

- Mach has achieved high portability. It typically takes less than three man-months to port Mach to a new hardware base.
- Mach's performance has been measured and compared to that of other operating systems. Initial indications are that its performance is generally competitive with other UNIX implementations such as SunOS, and markedly better in some cases (fork operation, large compilation). Its multiprocessor performance has also been measured and shown to be competitive with, for example, other Sequent and Encore operating systems. A key to Mach's performance gains is its implementation of virtual memory and its integration of virtual memory and communication.
- Mach Release 3, the kernelized version, is scheduled to be implemented by the end the summer and released by the end of the year.
- Mach has been widely distributed (to 200 institutions, 2/3 of which are corporations, 1/3 universities).
- Mach is serving as a platform for other distributed system research projects, such as Real-Time Mach, Camelot, Avalon, Strongbox, and Trusted Mach.

4. References

[Accetta 86]
Accetta, Mike, et al., "Mach: A New Kernel Foundation for UNIX Development," Computer Science Department, Carnegie Mellon University, Draft Paper, 1 May 1986.

[Baron 88]
Baron, Robert V., *MACH Kernel Interface Manual*, Computer Science Department, Carnegie Mellon University, Draft Paper, 15 February 1988.

[Cooper and Draves 87]
"C Threads," Computer Science Department, Carnegie Mellon University, Draft Paper, 2 March 1987.

[Draves 88]
Draves, Richard R., Michael B. Jones, and Mary R. Thompson, "MIG - The MACH Interface Generator," Computer Science Department, Carnegie Mellon University, Draft Paper, 26 February 1988.

[Jensen 85]
Jensen, E. Douglas, C. Douglass Locke, and Hideyuki Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of IEEE Real-Time Systems Symposium*, December 1985, 112-122.

[Jones and Rashid 86]
"Mach and Matchmaker: Kernel and Language Support for Object-Oriented Distributed Systems," *Proceedings of the 1st Annual ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), September 1986.*

Lehoczky 86]
Lehoczky, John P., Hide Tokuda, Lui Sha, and Dennis Cornhill, "ART: An Advanced Real-Time Technology Project," Computer Science Department, Carnegie Mellon University, Draft Paper, November 28.1986.

[Locke 86]
Locke, C. Douglass, *Best-Effort Decision Making for Real-Time Scheduling*, Ph.D. Dissertation, Carnegie Mellon University, 1986.

[Rashid 87a]
Rashid, Richard F., "From RIG to Accent to Mach: The Evolution of a Network Operating System," Computer Science Department, Carnegie Mellon University, 28 August 1987.

[Rashid 87b]
Rashid, Richard, et al., "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," *Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987.

[Sansom 86]
Sansom, Robert D., Daniel P. Julin, and Richard F. Rashid, "Extending a Capability Based System into a Network Environment," Technical Report CMU-CS-86-115, Computer Science Department, Carnegie Mellon University, 24 April 1986.

[Spector and Swedlow 88]
Spector, Alfred Z. and Kathryn R. Swedlow, editors, *Guide to the Camelot Distributed Transaction Facility: Release 1*, Computer Science Department, Mach/Camelot, Carnegie Mellon University, Draft of February 4, 1988.

[TCSEC 85]
"Department of Defense Trusted Computer System Evaluation Criteria," National Computer Security Center, DoD 5200.28-STD, December 1985.

[Tevanian 87]
Tevanian, Avadis, Jr., *Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach*, Ph.D. Thesis, Technical Report CMU-CS-88-106, Computer Science Department, Carnegie Mellon University, December 1987.

[TIS 88]
Trusted Information Systems, Inc., "Trusted Mach Presentation," Ellicott City, Maryland, 7 December 1988.

[Tokuda 87]
Tokuda, Hideyuki, James W. Wendorf, and Huay-Yong Wang, "Implementation of a Time-Driven Scheduler for Real-Time Operating Systems," *IEEE 8th Real-Time Systems Symposium*, December 1987.

[Tokuda 88]
Tokuda, Hideyuki, Makoto Kotera, and Clifford W. Mercer, "A Real-Time Monitor for a Distributed Real-Time Operating System," ACM SIGOPS/SIGPLAN Workshop on Distributed/Parallel Debugging.

[Tokuda and Kotera 88]
Tokuda, Hideyuki, and Makoto Kotera, "Scheduler 1-2-3: An Interactive Schedulability Analyzer for Real-Time Systems," Computer Science Department, Carnegie Mellon University, February 15, 1988.

[Yee 88]
Yee, Bennet S., J.D. Tygar, Alfred Z. Spector, "Strongbox: A Self-Securing Protection System for Distributed Programs," Technical Report CMU-CS-87-184, Computer Science Department, Carnegie Mellon University, 4 January 1988.

[Young 87]
Young, Michael, et al., "The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System," *Proceedings of the 11th Symposium on Operating Systems Principles*, November 1987.

V

1. Goals

The V distributed system project is a research project of David Cheriton at Stanford University. Its primary sponsor is DARPA, although several other organizations also provide some support.

V is a distributed operating system designed for a cluster of workstations interconnected by a high-performance network. It has been running at Stanford University since 1982. It currently runs on SUN and MicroVAX workstations, which are interconnected by a 10-megabit Ethernet. Its goals can be elaborated as follows:

- The V project strives for minimization of the kernel.
- The V project strives for high performance, in particular, high-performance interprocess communication.
- Originally, the target application domain for V was timesharing. It was envisioned that V would transform a cluster of workstations into a distributed system that would offer users the same resource and information sharing capabilities traditionally provided by a centralized timesharing system. This goal has been achieved. However, V's target application domain has been expanded to encompass both batch and real-time control applications, and therefore is essentially unrestricted.

2. Approach

The V distributed operating system consists of the following components:

- V kernel: The design of the V kernel is based on two key concepts. The first is that a kernel should be "minimal." Namely, it should implement an interconnection mechanism between applications and system services, but *not* the system services themselves. Thus, interprocess communication (IPC) lies at the core of the V kernel. The second key concept is that the kernel must satisfy the following "integrity constraint": the kernel cannot depend upon the correctness of anything outside of itself for its own correctness. If the kernel fails, then it must be either the kernel's fault or the hardware's fault. This integrity constraint limits the minimization (of the kernel) that can be achieved. Currently, the V kernel includes the following servers in addition to the IPC facility: a communication server (which implements the management component of IPC), a time server, a process server, a memory management server, and a device server. However, the design is periodically re-examined to determine whether further reduction of the kernel is possible.
- V system servers: These servers provide the traditional operating system services. They are implemented above the kernel, at the user process level, as multiprocess programs (based on lightweight processes). They are accessed through the V IPC mechanism. Current servers include a file server, a printer server, a display server, a pipe server, an Internet server, and a team server (which manages the execution of programs). Servers under development include a log server for optical disk storage and a time synchronization server.

Basic Abstractions

In the V literature, the V kernel is described as a "software backplane." Just as a hardware backplane provides slots, power, and communication, the V kernel provides address spaces, lightweight processes, and interprocess communication (in the form of message transactions). Thus, the basic abstractions of the V kernel are the following:

- Address space: The V kernel separates the conventional process abstraction into two components. The first component is the address space, which holds programs (and open files).
- Lightweight process: The second component of the process abstraction is the lightweight process, which is the locus of control within an executing program. Multiple lightweight processes may exist within an address space, and are referred to as a "team" of processes.
- Message transaction: Processes communicate via message transactions. In the basic scenario, a client *sends* a request message to a server, and then blocks (awaiting a response message). The server *receives* the request message, performs the requested service, and then *replies* to the client with a response message.

Problem-Oriented Shared Memory

The design of the V distributed operating system, and especially the V IPC facility, has been motivated by, and can also be justified by, the following line of reasoning:

- In order to transform a collection of autonomous computer systems into a *distributed* system, a distributed operating system must maintain "shared state" across the systems.
- The most appropriate abstraction for handling shared state is the shared memory paradigm, in which fetch and store operations apply *and fetch operations predominate*.
- To reduce network traffic and improve performance in the shared memory paradigm, a cache can be implemented at each node. But then the problem becomes maintaining the consistency of the data.
- It is recognized that many distributed system functions do not require absolute consistency. Therefore, "problem-oriented shared memory" is proposed as the paradigm upon which many distributed applications should be built. In problem-oriented shared memory, consistency requirements are relaxed, based on application-specific knowledge.
- The support of problem-oriented shared memory should be the driving force behind the design of a distributed operating system's IPC facility.

Interprocess Communication

V IPC is message-based. It has two distinguishing features. First, it is optimized for request-response behavior. Typically, a server runs as a dedicated process or team of processes. A client requests a service by sending a message to the server, and then waiting for the response. The request-response transaction (which is sometimes referred to as Remote Procedure Call (RPC) in the V literature) is considered fundamental in V. It directly implements the predominant fetch operation (typified by file read); namely, a client sends a request for data and receives the data in the server's corresponding response.

Second, V IPC supports multicast, both as a multi-destination delivery mechanism and as a binding (or logical addressing) mechanism. Multicast is considered fundamental to the implementation of problem-oriented shared memory, and has proved invaluable in the implementation of the V distributed operating system itself.

A transport level protocol, known as the Versatile Message Transaction Protocol (VMTP), has been developed to support V IPC. In addition to request-response and multicast transactions, VMTP also supports datagrams, forwarding, and streaming. In regard to streaming, it should be noted that VMTP, unlike other transport protocols, strives first for low delay, and then attempts to build high throughput capabilities (e.g., streaming) on top of the low delay foundation.

Storage Management

In V, an address space consists of ranges of addresses, called regions. The memory management system 1) binds regions to portions of open files (UIO objects), 2) manages physical memory as a cache for data from the open files, and 3) maintains the consistency of the cached data. The transfer of pages into the cache, as well as the mapping, is done on demand.

Uniform I/O (UIO) Interface

The V project has developed a uniform I/O interface called the UIO interface as its system-level I/O interface (as opposed to its application-level I/O interface, which is implemented by the run-time libraries). The UIO interface is based on an abstraction known as the UIO object, which corresponds to an open file in conventional systems. The UIO interface provides some support for record I/O, locking, atomic transactions, and replication. It further supports the notion of optional and exceptional (escape-mode) functionality.

Naming and Protection

V has a three-level naming system. At the highest level are character-string names, which are used for permanent objects such as files. At the next level are object identifiers, which are used for transient objects such as *open* files. At the lowest level are entity identifiers, which identify transport-level endpoints (such as processes or groups of processes).

Regarding protection, each process is encapsulated in an address space, and can communicate with other processes only via IPC.

Resource Management

In regard to processor scheduling, the kernel provides simple priority-based scheduling. That is, the kernel allocates the processor to the highest priority process. Above the kernel, the team server implements a higher level of scheduling. The team server can, for example, manipulate priorities to effect time-slicing.

Real-Time Support

V provides the following mechanisms, which offer some degree of support for real-time applications: datagram message transactions, prioritized message transmission and delivery, conditional message delivery (i.e., delivery only if the receiver is awaiting a message when the message arrives), priority-based scheduling, accurate time services, and memory-resident programs.

Reliability and Fault Tolerance

As previously mentioned, the UIO interface provides some support for replication and atomic transactions.

Programming Support Environment

The V distributed operating system offers programming support in the form of various run-time libraries. The libraries implement conventional programming interfaces such as Pascal I/O and C *stdio*. V also offers a set of system commands.

3. Accomplishments and Status

- V is being extended to run on shared memory multiprocessor machines. Targets include the DEC Firefly multiprocessor workstation and VMP, a shared memory multiprocessor machine designed and built at Stanford.

- Through its successful operation over a period of several years, V has proved the concept of building a distributed system on top of a minimal kernel.
- V has achieved high-performance communication, and in turn high-performance distributed applications. Through its performance and its emphasis on protocols, V has made an impact in the data communication field, as well as in the distributed operating systems field. Efforts are underway to promulgate some of its protocols, most notably VMTP, through the DoD data communication protocol standards process.
- V is serving as a base for continued research in distributed systems, as evidenced by the numerous publications listed below.

4. References

[Cheriton 84]
Cheriton, David R., "The V Kernel: A Software Base for Distributed Systems," *IEEE Software*, (April 1984), 19-42.

[Cheriton 86a]
Cheriton, David R., "Problem-oriented Shared Memory: A Decentralized Approach to Distributed System Design," *Proceedings of The 6th International Conference on Distributed Computing Systems*, May 1986, 190-197.

[Cheriton 86b]
Cheriton, David R., "VMTP: A Transport Protocol for the Next Generation of Communication Systems," *Proceedings of SIGCOMM 86*, August 1986, 406-415.

[Cheriton 87a]
Cheriton, David R., "UIO: A Uniform I/O System Interface," *ACM Transactions on Computer Systems 5*, 1 (February 1987), 12-46.

[Cheriton 87b]
Cheriton, David R., "Effective Use of Large RAM Diskless Workstations with the V Virtual Memory System," Computer Science Department, Stanford University, February 16, 1987.

[Cheriton 88a]
Cheriton, David R., "VMTP: Versatile Message Transaction Protocol," RFC 1045, SRI Network Information Center, February 1988.

[Cheriton 88b]
Cheriton, David R., "The V Distributed System," *Communications of the ACM 31*, 3 (March 1988), 314-333.

[Cheriton 88c]
Cheriton, David R., "Exploiting Recursion to Simplify RPC Communication Architectures," Computer Science Department, Stanford University, Draft Paper, March 21, 1988.

[Cheriton and Mann 88]
Cheriton, David R. and Timothy P. Mann, "Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance," to appear in *ACM Transactions on Computer Systems*, (1988).

[Cheriton and Roy 85]
Cheriton, David R. and Paul J. Roy, "Performance of the V Storage Server: A Preliminary Report," *Proceedings of the ACM Conference on Computer Science*, March 1985.

[Cheriton and Zwaenepoel 85]
Cheriton, David R. and Willy Zwaenepoel, "Distributed Process Groups in the V Kernel," *ACM Transactions on Computer Systems 3*, 2 (May 1985), 77-107.

[Finlayson and Cheriton 87]
Finlayson, Ross S. and David R. Cheriton, "Log Files: An Extended File Service

Exploiting Write-Once Storage," *Proceedings of the 11th Symposium on Operating System Principles*, November 1987, 139-148.

[Kanakia and Cheriton 87]
Kanakia, Hemant (Electrical Engineering Department) and David R. Cheriton (Computer Science Department), "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors," Stanford University, December 14, 1987.

[Tanenbaum and van Renesse 85]
Tanenbaum, Andrew S. and Robbert van Renesse, "Distributed Operating Systems," *Computing Surveys 17*, 4 (December 1985), 419-470.

[Theimer 85]
Theimer, Marvin M., Keith A. Lantz, and David R. Cheriton, "Preemptable Remote Execution Facilities for the V-System," *Proceedings of the 10th Symposium on Operating System Principles*, December 1985.

Distribution List for IDA Paper P-2142

NAME AND ADDRESS	**NUMBER OF COPIES**
Sponsor	
Lt Col Chuck Lillie SDIO Room 1E149 The Pentagon Washington, DC 20301-7100	6
LTC Jon Rindt SDIO Room 1E149 The Pentagon Washington, DC 20301-7100	2
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dr. Ashok Agrawala Department of Computer Science University of Maryland College Park, MD 20742	1
Dr. Jon Agre Science Center Rockwell International Corporation Mail Stop A24 1049 Camino Dos Rios Thousand Oaks, CA 91360	1
Lt Col Charles Anderson RADC/CO Griffiss AFB, NY 13440	1
CDR Rick Barbour Space & Naval Warfare Systems Command SPAWAR 324 Washington, DC 20363-5100	1
Ms. Donna Barker MS 202 Teledyne Brown 300 Sparkman Dr. Huntsville, AL 35807	1

NAME AND ADDRESS	NUMBER OF COPIES
Maj Brian Boesch DARPA Information Science and Technology Office 1400 Wilson Boulevard Arlington, VA 22209	1
Dr. James M. Boyle Mathmatics & Computer Science Division Argonne National Laboratory Building 221 Room C-219 9700 South Cass Avenue Argonne, IL 60439-4844	1
Dr. Jim Browne Deparment of Computer Sciences Taylor Hall 2.124 University of Texas Austin, TX 78712-1188	1
Mr. Ray Chen School of Information and Computer Science Georgia Institute of Technology Atlanta, GA 30332	1
Dr. David R. Cheriton Computer Science Department Bldg. 460, Room 422 Stanford University Stanford, CA 94305-6110	1
Mr. Ray Clark Department of Computer Science Carnegie Mellon University Pittsburgh, PA 15213-3890	1
Dr. Partha Dasgupta School of Information and Computer Science Georgia Institute of Technology Atlanta, GA 30332	1
Dr. Larry Dowdy Computer Science Department Vanderbilt University P.O. Box 1679, Station B Nashville, TN 37235	1

NAME AND ADDRESS	NUMBER OF COPIES
Michael J. Duffy GE Manager, Software Engineering Ground Systems Department 1787 Sentry Parkway West Blue Bell, PA 19422	1
Mr. Phil Hwang NSWC - White Oak Code U-33 Silver Spring, MD 20903-5000	1
Mr. E. Douglas Jensen Concurrent Computer Corporation 1 Technology Way Westford, MA 01886	1
Mr. Tom Lawrence RADC/COTD Air Force Systems Command Griffis AFB, NY 13441-5700	1
Dr. Richard J. LeBlanc, Jr. School of Information and Computer Science Georgia Institute of Technology Atlanta, GA 30332	1
Dr. John Lehoczky Department of Electrical and Computer Engineering Carnegie Mellon University Pittsburgh, PA 15213-3890	1
Dr. Doug Locke IBM Corporation Rout 17C Owego, NY 13827	1
Dr. Richard Metzger RADC/COTD Air Force Systems Command Griffis AFB, NY 13441-5700	1
Col John Morrison National Test Bed JPO Colorado Springs, CO 80912-5000	1

NAME AND ADDRESS	NUMBER OF COPIES
Dr. Isaac R. Nassi Vice President of Research Encore Computer Corporation 257 Cedar Hill Street Marlborough, MA 01752-3004	1
Dr. J. Duane Northcutt Department of Computer Science Carnegie Mellon University Pittsburgh, PA 15213-3890	1
Ms. Tricia Oberndorf Naval Air Development Center Code 7031 Warminster, PA 18974-5000	1
Lt Col Thomas J. Oldenburg ESD/XTS Hanscom AFB, MA 01731-5000	1
Dr. Bernard H. Paiewonsky Deputy for Advanced Technology SAF/AQH Room 4D977 Pentagon Washington, DC 20330-1000	1
Maj Mark Pullen DARPA Information Science and Technology Office 1400 Wilson Boulevara Arlington, VA 22209	1
Dr. Richard F. Rashid Department of Computer Science Carnegie Mellon University Pittsburgh, PA 15213-3890	1
Dr. John Salasin GTE 1700 Research Blvd. Rockville, MD 20850	1

NAME AND ADDRESS	NUMBER OF COPIES
Mr. Jim Sanderson Los Alamos National Lab Post Office Box 1663 MS K488 Los Alamos, NM 87545s	1
Dr. Richard E. Schantz BBN Laboratories Incorporated 10 Moulton Street Cambridge, MA 02238	1
Mr. Carl Schmiedekamp Naval Air Development Center Code 7033 Warminster, PA 18974-5000	1
Dr. Lui Sha Department of Computer Science and Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213-3890	1
Dr. Steve Squires DARPA Information Science and Technology Office 1400 Wilson Boulevard Arlington, VA 22209	1
Dr. Doyle Thomas USA SDC DASD-H-SB P.O. Box 1500 Huntsville, AL 35807	1
Dr. Hide Tokuda Department of Computer Science Carnegie Mellon University Pittsburgh, PA 15213-3890	1
Dr. Anand Tripathi Computer Science Department 136 Lind Hall University of Minnesota Minneapolis, MN 55455	1

NAME AND ADDRESS	NUMBER OF COPIES
Dr. Satish Tripathi Department of Computer Science University of Maryland College Park, MD 20742	1
Dr. Andre van Tilborg Computer Science Division, Code 1133 Office of Naval Research 800 N. Quincy St. Arlington, VA 22217-5000	1
Dr. Stephen T. Vinter BBN Laboratories Incorporated 10 Moulton Street Cambridge, MA 02238	1
Dr. Richard J. Waddell POET Office 1225 Jefferson Davis Hwy. Arlington, VA 22202	1
Dr. Cindy Williams MITRE Corporation MS T140 Burlington Road Bedford, MA 01730	1
CSED Review Panel	
Dr. Dan Alpert, Director Program in Science, Technology & Society University of Illinois Room 201 912-1/2 West Illinois Street Urbana, Illinois 61801	1
Dr. Barry W. Boehm TRW Defense Systems Group MS R2-1094 One Space Park Redondo Beach, CA 90278	1
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1

NAME AND ADDRESS	NUMBER OF COPIES
Dr. C.E. Hutchinson, Dean Thayer School of Engineering Dartmouth College Hanover, NH 03755	1
Mr. A.J. Jordano Manager, Systems & Software Engineering Headquarters Federal Systems Division 6600 Rockledge Dr. Bethesda, MD 20817	1
Mr. Robert K. Lehto Mainstay 302 Mill St. Occoquan, VA 22125	1
Dr. John M. Palms, Vice President Academic Affairs & Professor of Physics Emory University Atlanta, GA 30322	1
Mr. Oliver Selfridge 45 Percy Road Lexington, MA 02173	1
Mr. Keith Uncapher University of Southern California Olin Hall 330A University Park Los Angeles, CA 90089-1454	1
IDA	
General W.Y. Smith, HQ	1
Mr. Philip L. Major, HQ	1
Dr. Robert E. Roberts, HQ	1
Ms. Anne Douville, CSED	1
Dr. John F. Kramer, CSED	1
Mr. Terry Mayfield, CSED	4
Dr. Karen Gordon, CSED	30
Dr. Cathy Jo Linn, CSED	20
Dr. Cy Ardoin, CSED	1
Mr. Jim Baldo, CSED	1
Mr. Bill Brykczynski, CSED	1
Mr. Howard Cohen, CSED	1

NAME AND ADDRESS	NUMBER OF COPIES
Mr. Steve Edwards, CSED	1
Dr. Dennis Fife, CSED	1
Mr. Michael Kappel, CSED	1
Dr. Joe Linn, CSED	1
Dr. Reg Meeson, CSED	1
Dr. Jim Pennell, CSED	1
Mr. Kevin Rappoport, CSED	1
Dr. Eric Roskos, CSED	1
Mr. David Wheeler, CSED	1
Dr. Bob Winner, CSED	1
Ms. Katydean Price, CSED	2
IDA Control & Distribution Vault	3