(2)

IDA DOCUMENT D-1063

CONTRIBUTIONS TO THE OPERATING SYSTEMS STANDARDS WORKING GROUP OF THE NAVY NEXT GENERATION COMPUTER RESOURCES PROGRAM FOR FY 1989 - FY 1991

Karen D. Gordon, *Task Leader*

October 1991

Prepared for
Space and Naval Warfare Systems Command

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

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Group Reports

Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

Papers

Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

Documents

IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract MDA 903 89 C 0003 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.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1991	Final

4. TITLE AND SUBTITLE
Contributions to the Operating Systems Standards Working Group of the Navy Next Generation Computer Resources Program for FY1989 - FY1991

5. FUNDING NUMBERS
MDA 903 89 C 0003
Task Number T-D5-725

6. AUTHOR(S)
Karen D. Gordon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Defense Analyses (IDA)
1801 N. Beauregard St.
Alexandria, VA 22311-1772

8. PERFORMING ORGANIZATION REPORT NUMBER
IDA Document D-1063

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Space and Naval Warfare Systems Command
SPAWAR 231-2B1
Washington, DC 20363-5100

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, unlimited distribution: December 15, 1992.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (Maximum 200 words)

The Next Generation Computer Resources (NGCR) Program is a Navy standardization effort designed to fulfill the Navy's needs for standard computer resources while at the same time allowing it to take advantage of commercial products and investments and to field new technology more quickly and effectively. The program is centered around the selection and adoption of open systems standards in several areas, including backplanes, local area networks, operating system, programming support environments, graphics, and database management systems. IDA is providing support to the NGCR Program through participation in the NGCR Operating Systems Standards Working Group (OSSWG), a group chartered to establish commercially based operating system interface standards for use in Navy mission-critical computing systems in the mid-1990s and beyond. This document is a compilation of IDA contributions to the NGCR OSSWG for the time period from January 1989 through September 1991. It includes (1) a record of IDA support of a workshop held on behalf of the NGCR OSSWG, (2) copies of IDA contributions to NGCR OSSWG documents, and (3) copies of presentations given by IDA as a representative of the NGCR OSSWG.

14. SUBJECT TERMS	15. NUMBER OF PAGES
NGCR; OSSWG; POSIX; Operating System Interface; Open System Standards; Mission-critical Computing.	164
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

IDA DOCUMENT D-1063

CONTRIBUTIONS TO THE OPERATING SYSTEMS STANDARDS WORKING GROUP OF THE NAVY NEXT GENERATION COMPUTER RESOURCES PROGRAM FOR FY 1989 - FY 1991

Karen D. Gordon, *Task Leader*

October 1991

INSTITUTE FOR DEFENSE ANALYSES
Contract MDA 903 89 C 0003
Task T-D5-725

PREFACE

The U.S. Navy is operating a standardization effort known as the Next Generation Computer Resources (NGCR) Program. The NGCR Program is designed to fulfill the Navy's needs for standard computer resources while at the same time allowing it to take advantage of commercial products and investments and to field new technology more quickly and effectively. The program is centered around the selection and adoption of open system standards in several areas, including backplanes, local area networks, operating systems, programming support environments, graphics, and database management systems.

Since January 1989, IDA has been providing support to the NGCR Program in the area of operating systems. At that time, the U.S. Navy Space and Naval Warfare Systems Command formed the NGCR Operating Systems Standards Working Group (OSSWG), of which IDA is a member. The OSSWG is chartered to establish commercially-based operating system interface standards for use in Navy mission-critical computing systems in the mid-1990s and beyond.

This document, prepared in accordance with Task Order T-D5-725, is a compilation of *IDA contributions to the NGCR OSSWG*. It covers the time period from January 1989 though September 1991. It includes (1) a record of IDA support of a workshop held on behalf of the NGCR OSSWG, (2) copies of IDA contributions to NGCR OSSWG documents, and (3) copies of presentations given by IDA as a representative of the NGCR OSSWG.

CONTENTS

PART 1

Support of the

1989 Workshop on Operating Systems for Mission Critical Computing

This workshop was held in support of the NGCR OSSWG and was co-sponsored by the Office of Naval Technology, the Office of Naval Research, the Space and Naval Warfare Systems Command, the University of Maryland Department of Computer Science and Institute for Advanced Computer Studies, and IDA. Karen Gordon of IDA served as a Program Co-Chair for the workshop. In this part are copies of the following:

- Call for papers for the workshop
- Front material from the preliminary proceedings of the workshop
  - — Cover
  - — Inside cover
  - — Table of contents
- Program of the workshop

1989 WORKSHOP ON OPERATING SYSTEMS FOR MISSION CRITICAL COMPUTING

September 19-20, 1989
Greenbelt Marriott
Greenbelt, Maryland

Co-Sponsored by
Office of Naval Technology
Office of Naval Research
Space and Naval Warfare Systems Command
University of Maryland Department of Computer Science
University of Maryland Institute for Advanced Computer Studies
Institute for Defense Analyses

The U.S. Navy, the University of Maryland, and the Institute for Defense Analyses are sponsoring a workshop for the exposition of operating systems designed to meet mission critical computing requirements. Such requirements include, but are not limited to, real-time responsiveness, reliability, fault tolerance, and security. Operating systems that have explicitly taken into account the support of real-time Ada applications are of particular interest, due to the Department of Defense mandate on the use of Ada. Distributed operating systems are also of particular interest, due to the proliferation of mission critical computing facilities that are being realized through networks of interconnected computers.

Individuals or organizations wishing to present their operating system at this workshop are invited to submit 8 copies of a 5-10 page paper on the operating system to Karen Gordon by August 15, 1989. The paper should specify which mission critical computing requirements are addressed by the operating system. The paper should explain in what way, to what extent, and through what approaches and mechanisms the operating system attempts to meet the selected requirements. Descriptions of commercial operating systems, as well as of operating systems that exist in the context of research and development efforts, are solicited.

Participation at the Workshop will be limited to authors of accepted papers and representatives of the sponsoring organizations and other invited U.S. Government agencies. The accepted papers will be published as a Proceedings, which will be widely distributed within the Government and also made available to the public.

Submission Deadline: August 15, 1989
Acceptance Notification: August 29, 1989

General Chair:

Ashok Agrawala, Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742, (301) 454-4968, agrawala@mimsy.umd.edu

Advisory Committee Co-Chairs:

CDR Richard Barbour, Space and Naval Warfare Systems Command, Code 324A, Washington, DC 20363-5100, (202) 692-9207, barbour@nrl-css.arpa

Patricia Oberndorf, Naval Air Development Center, Code 7031, Warminster, PA 18974-5000, (215) 441-2737, tricia@nadc.arpa

Program Committee Co-Chairs:

Karen Gordon, Inst. for Defense Analyses, Alexandria, VA 22311, (703) 845-3591, gordon@ida.org

Phillip Hwang, Naval Surface Warfare Center, Code U33, Silver Spring, MD 20903-5000, (202) 394-1351, hwang@nswc-wo.arpa

PRELIMINARY PROCEEDINGS

1989 WORKSHOP
ON

OPERATING SYSTEMS
FOR
MISSION CRITICAL COMPUTING

SEPTEMBER 19 - 21, 1989

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND

Office of Naval Technology
Office of Naval Research
Space and Naval Systems Command
University of Maryland Department of Computer Science
University of Maryland Institute for Advanced Computer Studies
Institute for Defense Analyses

TABLE OF CONTENTS

1989 Workshop on Operating Systems
for Mission Critical Computing

Center of Adult Education
University of Maryland
College Park, Maryland

Tuesday, September 19, 1989

08:00 - 09:00 Registration

09:00 - 10:00 Introductory Session

- *Welcome*, Larry Davis, UMIACS
- *Office of Naval Technology Programs*, Jane Van Fossen, ONT
- *Office of Naval Research Perspectives*, Andre van Tilborg, ONR
- *Space and Naval Warfare Systems Command Programs*, Richard E. Barbour, SPAWAR
- *Overview of Mission Critical Computing*, Phillip Q. Hwang, NSWC

10:00 - 10:20 BREAK

10:20 - 12:20 Session I — Chair: Karen D. Gordon, IDA

- *The Spring Kernel: Operating System Support for Critical, Hard Real-Time Systems*, John A. Stankovic and Krithi Ramamritham, University of Massachusetts
- *The ARTS Distributed Real-Time Operating System*, Hide Tokuda, et al., Carnegie Mellon University
- *MARUTI, Hard Real-Time Operating System*, Ashok Agrawala, Satish Tripathi, and Olafur Gudmundsson, University of Maryland

12:20 - 13:40 LUNCH

13:40 - 15:40 Session II — Chair: Roger J. Martin, NIST

- *A Brief Overview of The ISIS Distributed Programming Toolkit and The Meta Distributed System*, Kenneth Birman and Keith Marzullo, Cornell University
- *V: A Foundation for Mission Critical Distributed Systems*, Kieran Harty and David R. Cheriton, Stanford University
- *Research on Imprecise Computations in Project QuartZ*, Jane W.S. Liu, et al., University of Illinois

15:40 - 16:00 BREAK

16:00 - 17:30 Discussion Session I — Chair: Andre van Tilborg, ONR

- *Mission Critical Operating Systems / Challenges*

17:30 - 19:00 DINNER

19:00 - 21:00 Evening Session I — Chair: C. Douglass Locke, IBM

- *Issues in Standardization*: Roger J. Martin, NIST, and Patricia Oberndorf, NADC

Wednesday, September 20, 1989

08:00 - 10:00 Session III — Chair: CDR Jane Van Fossen, ONT

- *Software Architecture and Prescriptive Interface for Real-Time OS*, Al Mok, University of Texas
- *The Distributed iRMX Operating System: A Real-Time Distributed Operating System*, Timothy G. Saponas and Roger B. Demuth, Intel Corporation
- *The Structure of the Clouds Distributed Operating System*, Partha Dasgupta and Richard J. LeBlanc, Jr., Georgia Institute of Technology

10:00 - 10:20 BREAK

10:20 - 12:20 Session IV — Chair: Stan Wilson, NRL

- *The StarLite Operating System*, Robert P. Cook, University of Virginia
- *Alpha: An Operating System for the Mission-Critical Integration and Operation of Large, Complex, Distributed Real-Time Systems*, E. Douglas Jensen, et al., Concurrent Computer Corporation
- *A Revolutionary System Architecture for Mission Critical Computing Systems*, J.C. Browne, University of Texas

12:20 - 13:40 LUNCH

13:40 - 15:40 Session V — Chair: CDR Rick Barbour, SPAWAR

- *DRAGON SLAYER/MELODY: Distributed Operating System Support for Mission Critical Computing*, Horst F. Wedde, et al., Wayne State University
- *SDEX/20 and 43RSS*, Barbara Haleen, et al., Unisys Corporation
- *Realtime Operating System for Secure, Mission Critical Avionics Systems*, Gary K. Miyahara and Cynthia L. Allyn, Hughes Aircraft Company

15:40 - 16:00 BREAK

16:00 - 17:30 Discussion Session II — Chair: Pat Watson, IBM

- *Issues in Application Development*: Susan Davidson, University of Pennsylvania, E. Douglas Jensen, Concurrent Computer Corporation, Brinkley Sprunt, Carnegie Mellon University, and John A. Stankovic, University of Massachusetts

17:30 - 19:00 DINNER

19:00 - 21:00 Evening Session II — Chair: Maj. Brian Boesch, DARPA

- *Mach: A Foundation for System Software*, Richard Rashid, Daniel Julin, Doug Orr, and Richard Sanzi, Carnegie Mellon University
- *Panel—Mission Critical Operating Systems / Initiatives*: Ira Goldstein, Open Software Foundation, Richard Rashid, Carnegie Mellon University, and Homayoon Tajalli, Trusted Information Systems

Thursday, September 21, 1989

08:00 - 10:00 Session VI — Chair: Patricia Oberndorf, NADC

- *Operating System Correctness is a Mission Critical Requirement*, William R. Bevier and Tad Taylor, Computational Logic, Inc.
- *The Secure Distributed Operating System—An Overview*, Rammohan Varadarajan, Joseph R. McEnerney, and D.G. Weber, Odyssey Research Associates
- *The BiiN Mission Critical Computer Architecture*, Fred J. Pollack and Kevin C. Kahn, BiiN

10:00 - 10:20 BREAK

10:20 - 12:20 Session VII — Chair: James G. Smith, ONR

- *A Real-Time Operating System for HARTS*, Dilip D. Kandlur, Daniel L. Kiskis, and Kang Shin, University of Michigan
- *Operating System Constructs for Managing Real-Time Software Complexity*, Prabha Gopinath, et al., Philips Laboratories
- *Choices for Mission Critical Computing*, Roy H. Campbell, John H. Hine and Vincent F. Russo, University of Illinois

12:20 - 13:40 LUNCH

13:40 - 15:40 Session VIII — Chair: John F. Kramer, IDA

- *Understanding the Needs of Ada Runtime Environments*, Mike Kamrad, Unisys Corporation
- *Real-Time Scheduling in Ada*, Mark Borger, et al., Software Engineering Institute
- *Distributed Ada Real-Time Kernel*, Roger Van Scoy, et al., Software Engineering Institute

15:40 - 16:00 BREAK

16:00 - 17:30 Discussion Session III — Chair: Ashok K. Agrawala, UoM

- *Mission Critical Operating Systems / Wrapup*

General Chair: **Ashok Agrawala, University of Maryland**
Program Co-Chairs: **Karen Gordon, Institute for Defense Analyses**
Phil Hwang, Naval Surface Warfare Center

PART 2

Contributions to the

NGCR OSSWG Available Technology Report[1]

In this part are copies of the following IDA contributions to the NGCR OSSWG Available Technology Report:

- BiiN/OS: Summary Description
- Clouds: Summary Description
- MARUTI: Summary Description
- Trusted Mach: Summary Description
- Clouds: Detailed Description
- *Cronus: Detailed Description*
- Mach: Detailed Description
- POSIX: Detailed Description
- V: Detailed Description
- Real-Time Operating System Technology

1. Version 1.3 of the *NGCR (Next Generation Computer Resources) OSSWG (Operating Systems Standards Working Group) Available Technology Report*, dated 09/14/1990, was published as a component of the *Operating Systems Standards Working Group (OSSWG) Next Generation Computer Resources (NGCR) Program First Annual Report—October 1990*, R. Bergman (editor), Technical Document 2101, Naval Ocean Systems Center, San Diego, California, October 1990.

BiiN/OS: Summary Description
Submitted by Karen Gordon (IDA)

BiiN/OS is the operating system of the Biin family of multiprocessor computers. BiiN/OS is designed to operate in distributed computing environment; multiple, geographically distributed Biin computers can be unified into a single "distributed" system through the Biin/OS. Biin/OS addresses fault tolerance, real-time, and security concerns. With respect to fault tolerance, it dynamically configures hardware modules to provide three levels of fault tolerance, which represent different tradeoffs between fault tolerance and performance. With respect to real-time, it provides traditional real-time application support, including preemptive, priority-based scheduling over a set of multiprocessors. With respect to security, it provides discretionary access control through a combination of capabilities and access control lists. In addition, BiiN/OS offers a UNIX-compatible operating system environment through its Biin Open Standard Interface Extension (BOSIX) tools. Points of contact: Steve Tolopka, Andy Crump, Biin, 2111 N.E. 25th Avenue, Hillsboro, Oregon, 97124-5961, (1-800) 252-2446.

Clouds: Summary Description
Submitted by Karen Gordon (IDA)

Clouds is a distributed operating system being developed at the Georgia Institute of Technology. It has received major funding from NSF, NASA, and RADC. Originally, the primary design goal of Clouds was the support of reliable, fault-tolerant distributed computing. The object/thread programming model (in which the traditional "process" is decomposed into an object, which serves as an abstraction of storage, and a thread, which serves as an abstraction of computation) was conceived as a means to an end, the end being reliable, fault-tolerant distributed computing. However, it has become an end in itself; the support and exploitation of the object/thread programming model 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. Points of contact: Rich LeBlanc and Partha Dasgupta, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA, 30332, rich@gatech.edu, partha@gatech.edu.

MARUTI: Summary Description
Submitted by Karen Gordon (IDA)

MARUTI is a real-time distributed operating system being developed at the University of Maryland. It has been funded, in part, by the U.S. Army Strategic Defense Command and the Office of Naval Research. MARUTI focuses on real-time and fault-tolerance requirements. It supports both in an object-oriented framework. With respect to real-time requirements, MARUTI supports guaranteed-service scheduling. That is, once it accepts a task, MARUTI guarantees that the timing constraints of the task will be met. It utilizes replication and consistency-control mechanisms to implement user-specified fault-tolerance constraints. Point of contact: Ashok Agrawala, Department of Computer Science, University of Maryland, College Park, MD, 20742, (301) 454-4968, agrawala@mimsy.umd.edu.

Trusted Mach: Summary Description
Submitted by Karen Gordon (IDA)

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. 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. Points of contact: Steve Walker, Marty Branstad, Trusted Information Systems, Inc., 3060 Washington Road (Route 97), Glenwood, MD, 21738.

Clouds: Detailed Description

Submitted by Karen Gordon (IDA) and Partha Dasgupta (Georgia Institute of Technology)

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.
- "Minimalist" philosophy for distributed operating system design and implementation. The Clouds operating system supports a minimal set of functions necessary to run a distributed system. The object/thread model provides a structured persistent memory, doing away with the need for long-term storage in the form of a file service and complicated I/O system. The operating system does not incorporate services such as printing, databases, and graphics, either, since these can be effectively implemented as user level applications. The kernel of the Clouds operating system is also minimal, in keeping with this philosophy.

1. Language Support Services

- Ada language support services.
- Support for other languages. The first language being supported on the Clouds system is C++. The language has been somewhat modified (by adding keywords) to support entry points, segmented data, persistent data, and permanent and

temporary memory allocation. The C++ programs define objects (and not processing). Support for single inheritance is complete, and support for multiple inheritance is being designed. The language defines the inheritance scheme, as Clouds does not define inheritance (but does have the mechanisms to implement inheritance and sharing effectively).

Ada, as well as some other languages, will be supported on Clouds at a later date.

2. Architecture Dependent Services

The Clouds researchers plan to support 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."

3. Capability and Security Services

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.

4. Data Base Services

Clouds does not provide database services. However, using persistent memory and true multitasking in objects, a variety of different database schemes can be implemented. These will be considered as applications on the Clouds system.

5. Data Interchange Services

Not incorporated into the operating system proper as an explicit service. However, it may be viewed as being supported by adherence to conventions.

6. Event and Error Management Services

Object-based error handling and event management will be provided by Clouds. However, the exact nature of these services is not completely defined at this time.

7. File Services

A conventional file can be viewed as a special case of a Clouds object, namely one with file data in its data space and file operations, such as read and write, which can be invoked by threads.

However, in the Clouds programming paradigm, the need for having files goes away. Programs do not need to store data in file-like entities, because they can keep the data in the (permanent) data space of objects.

Just as Clouds does not files, it does not provide user-level support for file (or disk) I/O. The system creates the illusion of a huge virtual memory space that is permanent, and thus the need for using disk storage form a programmer's viewpoint is eliminated.

8. Generalized I/O Services

9. Graphics Kernel Services

Not incorporated into the Clouds operating system proper. Since Clouds supports plug-in system-level objects, a graphics kernel can, in theory, be plugged into a Clouds operating system. Clouds, however, does not dictate the facilities by the graphics kernel.

10. LPOS to LPOS Communication Services

Clouds is a distributed operating system. Its kernel is replicated at each node that participates as part of a Clouds distributed system. The kernels implement Clouds IPC. System services, provided by system objects, are invoked using Clouds IPC.

11. Man-Machine Interface (MMI) Services

The Clouds researchers plan to provide an X-windows interface to Clouds.

12. Networks and Communications

- Interprocess communication. Clouds provides two modes of interprocess communication, both stemming from the Clouds paradigm of global persistent objects, which can be shared on a system-wide basis. 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.

13. Process Management Services

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.

14. Project Support Environment Services

15. Reliability, Adaptability, and Maintainability Services

Clouds 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.

Ongoing research is addressing fault tolerance using replicated data and computation.

16. Resource Management Services

- Storage management services. 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.

- General resource management services. 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.

17. Scheduling Services

18. Synchronization Services

19. System Initialization and Reinitialization Services

20. System Operator Services

21. Time Services

22. *Proprietary or Open*

Open, since it is a government-sponsored academic research project.

23. Qualification as a Standard

The Clouds project is directed at exploring a non-conventional methodology or paradigm for building operating systems. It is meant to demonstrate the utility and effectiveness of the new methodology. It is based on a minimalist philosophy, to allow for customization. Its focus, at this point in time, is still on the fundamental paradigm that it advocates. As the system matures, the researchers will likely explore higher level system services in more detail.

24. Platform Flexibility

Clouds defines a methodology that advocates unifying a distributed system using a set of global, persistent address spaces along with necessary structuring, naming, and consistency support. It couples long term storage with addressable memory, and decouples processing from storage. Clouds also demonstrates the following:

- This environment can be built using a structured, portable minimal kernel and plug-in system services.

- Most operating system services can be handled at the application level, allowing for customization.
- Management of shared persistent memory can be handled effectively in a distributed system.
- Consistency of persistent memory can be handled.
- Fault tolerance can be achieved and fine tuned through replicated objects and replicated computations.

The current prototype runs on a set of Sun 3/60 machines on an Ethernet. The design does not preclude any form of machine from multiprocessors to embedded systems or any form of networking.

25. 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: Detailed Description

Submitted by Karen Gordon (IDA) and Dale Brouhard (Naval Ocean Systems Center)

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

Cronus is an environment to support coherent integration of 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.

The Cronus approach is to introduce layers of software on top of constituent operating systems (or, in some cases, on bare hardware). Cronus 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.

Cronus supports heterogeneity by serving as a by-passable 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.

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.

1. Language Support Services

- Ada language support services. Programming support is now being extended to Ada, as noted below.
- Support for other languages. The fundamental assumption underlying Cronus programming support is that large-scale applications will be developed in accordance with the object model, just as Cronus 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.

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.

2. Architecture Dependent Services

3. Capability and Security Services

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.

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.

4. Data Base Services

Distributed databases are being addressed in an ongoing research project, the Cronus Distributed DBMS Project. The primary focus of the project is to design a distributed database management system for the Cronus environment. The purpose of the DBMS integration activity was to make a database system available from within Cronus in the near-term while the distributed DBMS design was in progress.

The near-term goals that have been satisfied include:

- Provide host-independent access by any Cronus client to the database system (currently Informix).
- Provide remote clients with the full functionality of the database system, including *multistatement transactions* and schema manipulation operations.
- Support access control to the database system at the database, table and column level.
- Support client/database interactions that are optimum for the different types of uses (e.g., highly interactive vs embedded) using SQL.

5. Data Interchange Services

Cronus uses the technique of canonical data representation to solve the problem of data interchange in a heterogeneous computing environment. Programs process data in formats directly supported by the systems on which they are implemented. When data is transferred to another network component, it is encoded into a canonical form using appropriate conversion routines. The reverse process takes place on the receiving end. Cronus takes non-procedural specifications of a new object type, and automatically generates code for data conversion between canonical and system-specific data representations.

6. Event and Error Management Services

Cronus routines that detect errors generally signal a failure by returning a special value distinguished from the set of normal return values. Routines returning numeric results can return ERROR, and routines that return pointer can return NULL. Before returning, these routines generally set the "ErrorBlock"—a global structure in the program that records error conditions. Then, the calling routine has the option of attempting error recovery action using the information it find in the ErrorBlock, resetting the ErrorBlock with its own interpretation of the error, or simply returning the error indication provided by the lower-level routine.

7.Files Services

Cronus provides a file system for storing information just as do other operating systems. Cronus files are objects, so they are accessible through the same object-oriented, location independent IPC facility as are other Cronus system entities. Files in the Cronus files system are accessible from any and all hosts in the Cronus cluster automatically. Cronus will locate a file and direct operations to that file's object manager transparently, hiding the distributed nature of the filesystem, and providing an interface to the application program or user that is simple and powerful.

Though, different file types are implemented as different object types, the object-oriented nature of Cronus allows all of these file types to respond to a common set of operations through the mechanism of inheritance. Currently, two Cronus file types are available:

- COS files (constituent operating systems files). Ordinary local host operating system files that have been made into objects and are accessible from anywhere within Cronus.
- Reliable files. Characterized by special facilities for synchronizing access by multiple simultaneous readers and writers, by enhanced read and write operations that can simplify application programs, and by enhanced survivability in the face of system failures.

8. Generalized I/O Services

Devices, such as line-printer, tape-driver, or terminal, are integrated into the Cronus system as sub-types of a generalized I/O object, which supports a generalized set of I/O operations. File- like interfaces for device I/O are supported for most devices.

9. Graphics Kernel Services

10. LPOS to LPOS Communications Services

Cronus Kernels communicate with one another using the Cronus Peer to Peer protocol. Reliable delivery of datagrams is guaranteed through the use of TCP as the transport mechanism. The Peer to Peer Protocol includes a specification for establishing TCP links between Kernels and for closing them down. A provision is also included to send low effort datagrams and broadcast/multicast messages between Kernels using UDP datagrams.

11. Man-Machine Interface (MMI) Services

The nature of a particular native operating system determines what user interface is supported.

12. Networks and Communications

- Network control and status. The monitoring and control system (MCS) for Cronus includes monitoring and control of hosts and of the Cronus functions on these hosts, of the network substrate and of gateways.

- 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.

13. Process Management Services

Since Cronus is based on the object model, the basic abstractions are objects and operation invocations. To implement the object model, the Cronus 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.

14. Project Support Environment Services

The Tropic (Transportable Operation Interface for Cronus) program allows a user to invoke arbitrary operations on Cronus objects (e.g., interactive debugging of an object manager on the target system).

15. Reliability, Adaptability and Maintainability Services

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.

Two replication strategies are now available. The first, referred to as version voting, mandates application-specified vote quorums to perform read and update operations. Version vectors are used to detect and correct inconsistencies. The second replication strategy, referred to as weak consistency, it that provided by precious versions of the Cronus manager development tools. BBN will be looking into other algorithms for replication. 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.

16. Resource Management Services

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.

17. Scheduling Services

Cronus object managers use a coroutine package which provides a C programming interface for priority-ordered, non-preemptive multiple threads of execution within a single constituent operating system process, along with mechanisms for synchronization and mutual exclusion of critical sections.

18. Synchronization Services

Cronus provides concurrency control for sensitive regions via the Start Concur and EndConcur routines. These are used in an object manager to bracket critical sections of code and provide concurrency control for accesses to the object database. This permits multiple operations to access the same object without fear of creating any inconsistencies in that object or it object database.

Semaphores are provided to control access to critical data structures. Since Cronus object managers may consist of multiple threads within a single process, semaphores can be used when data whose integrity must be insured may be shared by one or more tasks. The semaphore package causes threads to sleep when they are waiting to enter critical sections, and be awakened when the resource they are awaiting is available.

19. System Initialization and Reinitialization Services

Cronus object managers are asynchronous independent processes which are started by the system when it boots or by other processes (users).

20. System Operator Services

The Cronus operator commands are used by system operators to perform system maintenance tasks on a Cronus cluster. Currently the Cronus operator commands offer the following services: (1) start the Cronus kernel and managers, (2) examine contents of an object database, (3) examine contents of an object, (4) repair incorrectly replicated directories, (5) repair incorrectly replicated objects, (6) initialize Cronus file system, (7) initialize Cronus UNO file, (8) manipulate Cronus Broadcast Repeater, (9) examine circular log files, (10) stop Cronus on a particular host.

21. Time Services

Via constituent (i.e., native) operating systems.

22. Proprietary or Open

Sponsored by Rome Air Development Center (RADC).

23. Qualification as a Standard

Essentially, the Cronus approach to dealing with heterogeneity is to introduce a layer of "standardization," in the form of the Cronus environment, between constituent operating systems and application programs. The issue is the utility, effectiveness, appeal, and acceptability of the Cronus environment. To date, Cronus has received only isolated support outside of BBN.

24. Platform Flexibility

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.

Cronus implementations exist for the following systems: DEC VAX with VMS, Ultrix, and BSD Unix; SUN 2,3,4 and Sun 386i 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.

25. References

[BBN 88a] BBN Laboratories Incorporated, *Operator's Reference Manual, Release 1.3*, September 15, 1988.

[BBN 88b] BBN Laboratories Incorporated, *Programmer's Reference Manual, Release 1.3*, September 15, 1988.

[BBN 88c] BBN Laboratories Incorporated, *Tutorial Documents, Release 1.3*, September 15,1988.

[BBN 88d] BBN Laboratories Incorporated, *User's Reference Manual, Release 1.3*, September 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.

[Vinter 88] Vinter, Stephen T., et al., "The Cronus Distributed DBMS Project: Program Specification," Report No. 6854, BBN Laboratories Incorporated, June 1988.

Mach: Detailed Description

Submitted by Karen Gordon (IDA) and Dale Brouhard (Naval Ocean Systems Center)

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).

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.

1. Language Support Services

- Ada language support services. Mach threads (light-weight processes) could be used to handle Ada tasking.
- Support for other languages. An interface specification language, MIG (Mach Interface Generator), has been developed for Mach. MIG generates C or

Common Lisp RPC stubs.

2. Architecture Dependent Services

3. Capability and Security Services

- 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.

4. Data Base Services

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.

5. Data Interchange Services

Matchmaker is an interface specification language for use with existing programming languages. Differences in type representation by various programming languages within each machine are handled by Matchmaker. Data representation issues across machine boundaries are handled through message server processes. Byte reordering and machine specific conversions are performed by the message servers with the responsibility for conversion always resting with the receiving host.

6. Event and Error Management Services

Mach utilizes message passing for the invocation of exception handlers. When a thread raises an exception, a message is sent to its thread exception port to notify its error handler, which executes in a separate thread. If no handler exists or the handler fails to recover the exception, the message is forwarded to the exception port of the task in which the exception-incurring thread exists.

A debugger can intercept unhandled exceptions for all threads in a task by attaching itself to the task exception port. This enables a debugger to coexist with error handlers, in that the debugger is aware only of those exceptions not handled by an error handler.

7. File Services

The current vision is for Mach to be based on a minimal kernel upon which multiple file systems can be built. At this point, the kernelization is not complete, and the UNIX file system (4.3BSD)functionality is still embedded in Mach kernel code.

8. Generalized I/O Services

Currently, the UNIX I/O interface is used with low-level device drivers residing in the Mach kernel. Virtual-memory based file-mapping replaces buffer management in the standard I/O libraries

9. Graphics Kernel Services

Integration of IPC and virtual memory management (e.g., copy-on-write) provide some support for a graphics kernel.

10. LPOS to LPOS Communication Services

11. Man-Machine Interface (MMI) Services

X-Windows is used as a basis for the MMI along with the UNIX shells.

12. Networks and Communications

- 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. The Netmsgserver passes all the Mach IPC message between machines. It also provides network wide port register and lookup functions.

  The Environment Manager can register or look up ports or named strings but does not communicate with other Environment Managers.

13. Process Management Services

- Basic abstractions. Mach divides the process abstraction into two orthogonal abstractions: the task and the thread. A task is a collection of system resources. These include a virtual address space and a set of port rights. The thread is the basic unit of computation; it is the specification of an execution state within a task. Mach allows multiple threads to execute within a single task.

  Operations on tasks and threads are invoked by sending a message to a port representing the task or thread. Threads maybe be created, destroyed, suspended, and resumed.

  Tasks are related to one another in a tree structure by task-creation operations. Regions of virtual memory may be marked for future child tasks as either inheritable read/write, copy-on-write, or as neither.

14. Project Support Environment Services

15. Reliability, Adaptability, and Maintainability Services

- 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.

16. Resource Management Services

- 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. Not ably, 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.

17. Scheduling Services

- 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.

18. Synchronization Services

Mach provides constructs for protection of critical regions and synchronization. Lock and unlock primitives are used with mutex variable to provide mutual exclusion. Wait and signal primitive are used with condition variables to provide synchronization.

At a higher level, Matchmaker and MIG provide a synchronous/asynchronous RPC interface.

19. System Initialization and Reinitialization Services

20. System Operator Services

Based on UNIX.

21. Time Services

Based on UNIX.

22. Proprietary or Open

Mach is open. It has been widely distributed (to over 200 institutions, 2/3 of which are corporations, 1/3 universities).

23. Qualification as a Standard

In initiating the Mach project, DARPA aimed to capitalize on the de facto standard status of UNIX. Mach's UNIX compatibility is fundamental to its success and popularity. The Mach project is meant to rebuild the core of UNIX while retaining its external interfaces.

DARPA is participating in the various UNIX standardization efforts, such as POSIX and OSF. It definitely wants to exert its influence and make Mach a dominant operating system. Inasmuch as UNIX qualifies as a standard, Mach also does.

24. *Platform* Flexibility

As pointed out in the introduction, 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.

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.

25. 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.

[Leblanc and Miller 88] Leblanc, Thomas J. and Barton P. Miller, editors, "Summary of ACM Workshop on Parallel and Distributed Debugging," held May 5-6, 1988, University of Wisconsin, Madison, Wisconsin, *ACM Operating Systems Review 22*, 4 (October 1988), 7-19.

[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.

POSIX: Detailed Description

Submitted by Doug Locke (IBM) and Karen Gordon (IDA)

The IEEE Portable Operating System (POSIX) interface standard is based on earlier UNIX operating system interfaces. This standard defines an application program interface to an underlying set of operating system functions; it does not specify the structure, functions, or performance of the underlying operating system beyond the specific functionality visible at the application program interface level.

This standards committee, IEEE P1003, has created an initial version of the interface standard which completed balloting in August, 1988, and is therefore identified as IEEE 1003.1-1988. In addition, this committee includes a number of smaller working groups which are expected to present extensions or application-specific interfaces for this standard for official balloting before 6/90; draft versions of these extensions are currently available.

In this section, we describe the characteristics of the POSIX interface standard, including information from current draft extensions of the 1003.2 (Shells and System Utilities) working group, the 1003.4 Real-Time Extensions working group (Draft 7), and the 1003.5 Ada Bindings working group, using the framework of the NGCR-OSSWG Reference Model.

1. Language Support Services

The POSIX standard is currently expressed in terms of the C language, but is planned for language-independent revision for future releases.

The POSIX Ada Binding working group, IEEE P1003.5 is currently defining an interface to POSIX from the Ada language; this interface is expected to enter formal balloting by 6/90. The Ada language interface will provide access to all POSIX functions; initially, the Ada bindings working group is targeting 1003.1-1988, but plans to target the real-time extensions from 1003.4 immediately following the initial 1003.1-1988 work. Similarly, the POSIX interface must support all Ada functionality.

As with many commercial operating system interfaces, the POSIX process semantic model currently provides a poor match to Ada tasking, although the provision (in the POSIX 1003.4 Real-Time Extensions) of asynchronous I/O is expected to significantly improve the ability of Ada implementations to support Ada tasking within the process model by removing opportunities for blocking system calls issued by one Ada task to block the entire Ada program in the POSIX process. Beyond the POSIX process model, a light-weight concurrency mechanism within a POSIX process (i.e., threads) is also under active consideration by the 1003.4 Real-Time Extensions working group; if

accepted, its presence is expected to significantly improve the semantic match between POSIX and Ada tasking, greatly enhancing the suitability of POSIX to handle Ada tasking (see Process Services description). In addition, the timer and event mechanisms (also defined in the 1003.4 Real-Time Extensions) can be used effectively to implement the Ada delay and exception mechanisms.

2. Architecture Dependent Services

By its nature, the POSIX interface definition is architecture independent; this has been a primary requirement during its definition, and one which has been rigorously enforced by the many processor vendors participating in its definition. Beyond the POSIX standard, conforming implementations are expected to provide implementation-defined services to support specific architectures, but they are constrained to do this in ways which will not violate the POSIX standard.

3. Capability and Security Services

The current standard specifies file-level permissions for read, write and execute by the owner, a specified group of users, or all users. Further security facilities are planned and are under consideration by a POSIX Security working group 1003.6. Prior to the definition of such security extensions, conforming implementations are free to extend the standard to provide appropriate discretionary and mandatory access controls required to conform to NSCS Orange Book security requirements.

4. Data Base Services

While POSIX does not specifically include database support at the application interface, it does provide functions which may be used as building blocks in the construction of database functions. For example, the POSIX 1003.4 Real-Time Extensions draft defines extensions which support database functions, including real-time files (e.g., contiguous files for which application response produces predictable temporal and I/O performance), semaphores to allow data consistency controls, and asynchronous I/O functions to allow concurrent multiple I/O operations and computations within a POSIX process. For further information, see File Service and Synchronization Service descriptions.

5. Data Interchange Services

POSIX provides a "pipe" facility for serial communications between processes using the standard file service interfaces. For this facility, each process opens its end of the pipe, which then transmits characters between them.

Additionally, the POSIX 1003.4 Real-Time Extension provides support for message passing as a form of interprocess communication (shared memory is an alternate

form, which is also supported.) It views the capability of passing messages with both high and deterministic performance as being crucial in real-time systems. It implements message passing through "message queue special files," i.e, objects named within the file system (although this definition places message queues in the file system name space, this does not imply that the function need be handled by the file management portions of the operating system; neither does it necessarily imply a FIFO queueing discipline.). Message queue special files can be opened for use by multiple sending and receiving processes.

Functions

The POSIX 1003.4 message passing facilities provide the following functions:

- Creating a message queue special file.
- Opening and closing a specified message queue special file.
- Sending a message to a specified message queue special file. The message to be sent is identified by a pointer (to a buffer in which the message is held) and a length. If the length is zero, then the pointer itself is the message, which facilitates an optimization for very short messages. This optimization entails passing the contents of the pointer to the receiver as the event value field in the asynchronous event notification associated with the message. This is an optimization in that the asynchronous event notification not only notifies the receiver of a message receipt, but also passes the message as part of the notification. It should be noted that the pointer's worth of information can indeed be a pointer - to a buffer in shared memory, for example. The sender can specify that the message is to be sent either synchronously, in which case the sender blocks until the message is delivered to the receiver, or asynchronously, in which case the sender does not block. In the asynchronous case, the sender can further specify whether or not asynchronous event notification is to occur upon receipt. If the message is held in a buffer (as opposed to a very short message entirely contained in a buffer pointer), then the sender can specify how the contents of the buffer is to be transferred -- by transferring control of the buffer to the receiver, by copying the contents of the buffer into an intermediate system buffer, or by granting access to the buffer to the receiver so that the receiver can copy the contents. Finally, the sender can associate a "type" with the message, which the receiver can use to selectively receive messages. The type can be used to prioritize messages, for example.

- Sending a message to a specified list of message queue special files, thus providing a multicast capability.
- Receiving a message from a specified message queue special file. The receiver can specify that the message is to be received asynchronously, via an asynchronous event notification, or synchronously. If synchronous receipt is specified, then the receiver can further specify that the receipt is to be blocking (the receiver blocks until a message is available) or conditional (the receiver does not block, but only receives a message if one is available). The "type" field can be used to used to selectively receive messages in one of three ways: (1) FIFO delivery, (2) prioritized delivery, with FIFO for messages of equal priority (i.e., type), and (3) FIFO delivery of a specified type.
- Allocating (by the sender) and freeing (by the receiver) a system-provided "message buffer" to hold the message, thus minimizing message copying.
- Setting the values of and getting the values of attributes of a specified message queue special file. Attributes include the maximum number of messages, maximum number of bytes, whether or not to "wrap" new messages over old ones, etc.
- Getting the status of a specified message queue special file.

6. Event and Error Management Services

A "signal" mechanism is provided which allows operating system or application defined events to be sent to a process. The process may elect to handle the signal asynchronously or to ignore it, in which case the process will be terminated, along with its children. If two signals arrive before the first is handled, the first will be lost; the signal mechanism is therefore considered unreliable.

The POSIX 1003.4 Real-Time Extension provides an additional service; an event mechanism is available providing for delivery of reliable asynchronous event notifications.

POSIX 1003.4 views asynchronous event notification as being essential in real-time systems. It strives to provide a general purpose, uniform, reliable interface that has both determinism and high performance. Its asynchronous event notification facilities consist of the following: (1) event definition data structure, (2) event trap routine function prototype definition, and (3) the functions cited below. In defining an event, a user specifies an event trap routine, an application-defined event value to be passed to the event trap routine identifying the source of the event, the event class (i.e., a grouping of

related events, including a priority in case of multiple event occurrences) within which the event trap routine executes, and the event class mask to be in effect during execution of the event trap routine.

Examples of predefined asynchronous events defined in the POSIX 1003.4 Real-Time Extensions standard include asynchronous I/O completion, timer expiration, message arrival, as well as user-defined events.

Functions

The POSIX 1003.4 asynchronous event notification facilities provide the following functions:

- Changing or examining the event class mask of the invoking process. Event classes that are included in the mask are blocked from being delivered via asynchronous event trap routines. Events are queued if they occur when masked.
- Waiting for asynchronous event notifications for specified event classes, in one of two modes. In the first mode, the event processing is handled by the associated event trap routines; that is, the invoker is "enabling" the delivery of asynchronous event notifications to their respective trap routines. In the second mode, referred to as polling, the event processing is handled in-line; that is, the event notification (i.e., event class and application-defined event value) is delivered to the caller as part of the return from the invocation of the wait. The caller can specify that the wait be one of the following: (1) zero, in which case the caller does not wait, but only enables pending event notifications to be delivered, (2) indefinite, or (3) subject to a specified timeout period.
- Causing a specified application-defined event to be raised for the invoking process.
- Changing the number of queue entries to be used to hold events which have been raised but not yet delivered to the invoking process.
- Achieving reliable exits from event trap routines via non- local jumps.
- Associating a specified signal with a specified event class. This is a desirable, but not mandatory, capability that enables a deterministic delivery order to be achieved for signals.

Notes

- Event classes establish a prioritization for event notification delivery, in the sense that if event notifications of different event classes are queued, then they are

dequeued in order of event class, and in FIFO order for event notifications of equal event class.

- The POSIX 1003.4 asynchronous event notification facilities are purposefully not intended to provide interprocess communication, although they are used as a delivery mechanism by the IPC mechanism described under Data Interchange Services.

7. File Services

The POSIX file system consists of a hierarchically organized set of files. The hierarchy consists of a set of file directories which in turn contain pointers either to other directories or to individual files. File descriptors may be entered into more than one directory; a file is not deleted until it is removed from the last directory in which it is entered. In addition, the file system may contain "special" files which are visible in a directory, but may or may not have representations in permanent storage and do not imply that their associated functions are performed by the file management portions of the operating system. They are used for operations which cross process boundaries, such as semaphores and shared memory.

The POSIX 1003.4 Real-Time Extensions define three extensions to the file services for use in developing real-time and database applications. These services are real-time files, asynchronous I/O services, and synchronized I/O services.

7.1 Real-time Files

The POSIX 1003.4 Real-Time Extensions draft views the capability of performing I/O operations with both deterministic and high performance as being crucial in real-time systems. It recognizes that contiguous files are a traditional mechanism for providing deterministic high performance I/O, since most real-time systems utilize rotating magnetic disks as their file storage media, but rather than provide a specific interface to contiguous files, it provides a more general interface to "real-time files." Of course, an implementation may choose to implement real-time files using contiguous files, but it is not forced to do so. An implementation is free to take advantage of advanced or non-traditional media that can provide deterministic high performance without relying on contiguity, within a framework provided by the POSIX 1003.4 real-time file facilities.

The approach that POSIX 1003.4 takes to real-time file support is to make some critical (performance-related) attributes of the operating system's implementation of files and I/O not only visible to applications but also to some extent application controllable (i.e., at least "influenceable," through hints related to characteristics of the application, which the operating system can take into account).

Deterministic high performance means predictable (i.e, time bounded) and very short delay times.

Functions

The POSIX 1003.4 real-time file facilities provide the following functions:

- Creating a real-time file. In doing so, a process communicates to the system what it perceives as being desirable attributes of the real-time file, and it receives in return information on the actual attributes of the file as created by the system. As stated in the POSIX 1003.4 document [POSIX 1003.4 draft 7, p. 167], "A conforming application should compare all of the requested and returned attributes to ensure a sufficient set have been honored."
- Communicating to the system desirable attributes of a specified (previously created) real-time file. Again, the specification of desirable attributes is just a request to the system. The process receives in return information on the actual attributes, and must decide how to proceed based on that information.
- Getting the actual attributes of a specified real-time file or of real-time files of a specified file system.
- Obtaining a suitably aligned buffer of a specified size either from a specified data area or from the system.

Notes

- Performance-related implementation attributes include the following: (1) amount of data to be transferred in real-time I/O operations, (2) space reserved for a file or file system, (3) whether or not file is to be extended, and (4) size of extents.
- Hints, or advisory information, that an application can offer the operating system include the following: (1) file allocation should be optimized for sequential access, which can be taken to mean that contiguous allocation is desirable, (2) remaining file space should be zeroed upon truncation of the file, (3) reaccess of currently accessed data is unlikely, so a least recently used (LRU) caching policy might not be advantageous. (4) read-ahead might not be advantageous, and (5) application is doing its own caching, so system caching might not be advantageous.

7.2 Asynchronous I/O Services

The POSIX 1003.4 Real-Time Extensions provide an additional capability to perform file I/O asynchronously, allowing processes to perform multiple concurrent I/O

operations concurrently with computations on I/O data.

Functions

The POSIX 1003.4 asynchronous I/O facilities provide the following functions:

- Asynchronously reading and writing a specified file. Control returns to the invoker when the read or write request has been initiated, or, at a minimum, when it has been queued to occur. The invoker can define an event and associate it with the asynchronous I/O operation, in which case the event is raised upon completion of the operation. Alternatively, the invoker can poll for completion by checking a field in the control block specified for the I/O operation. The invoker can also specify an asynchronous I/O operation priority to be used in determining the order of execution of asynchronous I/O operations with respect to one another. The significance of the priority is implementation- specific.
- Initiating a list of I/O requests with a single system call.
- Cancelling a specified asynchronous I/O request; or, cancelling all asynchronous I/O requests to a specified file.

7.3 Synchronized I/O Services

In addition, the POSIX 1003.4 Real-Time Extensions working group recognizes that some applications require assurance of I/O completion, particularly in database applications. It views the capability of receiving such assurance as being vital in real-time systems. The POSIX 1003.4 Real-Time Extensions draft refers to I/O that is to be done with assurance of completion as "synchronized I/O."

Two types of synchronization are defined in POSIX 1003.4:

- Synchronized I/O data integrity completion. Completion is defined to occur for reads when data becomes available to the reading process. It is defined to occur for writes when both the data and the file system information necessary for retrieval of the data have been successfully transferred.
- Synchronized I/O file integrity completion: completion is defined to occur when, in addition to the above, all file system information relevant to the I/O operation has been successfully transferred.

Data is said to be successfully transferred "when the associated I/O peripheral in some implementation-defined fashion assures that all data is readable on any subsequent open of the file in the absence of a failure of the physical storage medium" [POSIX Draft 7, p.11].

Functions

The POSIX 1003.4 synchronized I/O facilities provide the following functions:

- Specifying that I/O completion for a specified file is to be (re-)defined as either synchronized I/O data integrity completion or as synchronized I/O file integrity completion. A process can make this specification for a file that is being opened (as part of the POSIX open() function) or for one that is already open (via the POSIX fcntl() function).
- Requesting that all outstanding I/O requests for a specified file are to be "completed" in accordance with the definition of either synchronized I/O data integrity completion or as synchronized I/O file integrity completion. The request can be either synchronous (i.e., control returns to the invoking process upon the completion of all the outstanding I/O operations) or asynchronous (i.e., control returns when the request is queued). In the asynchronous case, a specified asynchronous event notification will occur upon completion.

8. Generalized I/O Services

Device I/O in POSIX is handled in the same way as file I/O. Device access is made through the normal file system operations, although additional primitives are available to control devices.

9. Graphics Kernel Services

See the Man-Machine Interface (MMI) Services description.

10. LPOS to LPOS Communications Services

At the POSIX application interface, no functions are provided for inter-LPOS services. Conforming implementations are expected to provide such services transparently using normal POSIX functions, or using implementation-defined extensions.

11. Man-Machine Interface (MMI) Services

The POSIX 1003.2 Shells and Utilities working group is currently balloting a draft standard which provides for a tty-style command interface at the man-machine interface level which is virtually identical to the UNIX interface. The provision of functions to control user consoles at the application functional interface makes it possible to implement virtually any man-machine interface desired; typical examples provided by conforming implementations might include X-windows (from MIT), MOTIF (from the Open Software Foundation), and Open Look (from Unix International). A separate IEEE standard committee called 1102 is defining a standard with respect to a graphics

interface, based on X-windows.

12. Networks and Communications

POSIX itself does not (now) address communications directly. Conforming POSIX implementations are free to provide any level of communications control desired, including transparent distributed facilities in which the existence of multiple processors is not visible to the application.

13. Process Management Services

The POSIX 1003.1-1988 standard offers "heavy-weight" process concurrency. Each process has a separate address space, and shares file descriptors and other control structures with its parent. Operating system primitives to manage processes are very easy to use (i.e., fork() and exec()).

A proposal to add "light-weight" a concurrency model (called "threads") within a POSIX process is under active consideration by the POSIX 1003.4 Real-Time Extension working group. This would provide for multiple threads of control to exist within processes, including mutual exclusion primitives (e.g., mutex). These threads would carry very little state information (i.e., stack pointers and registers) which would make them extremely "light weight" and thus allow extremely fast implementations. They would be priority scheduled in the same way as POSIX processes (see Scheduling Services), using the same set of possible priorities.

14. Project Support Environment Services

The POSIX interface description does not explicitly describe project support environment services.

15. Reliability, Adaptability and Maintainability Services

The POSIX interface standard includes extensive error checking for every service, defining a complete set of error returns when errors are detected, as well as an asynchronous error facility (i.e., signals and events). Actual management of error conditions, including reconfiguration, is intended to be handled by applications using these error indications with standard POSIX functions.

16. Resource Management Services

The POSIX 1003.2 Shells and Utilities working group includes user-accessible memory management services (e.g., malloc()) which provide for memory allocation within the process memory space. The POSIX process model provides for the process memory spaces to be mutually disjoint; thus, the memory management services need not

be included in the POSIX kernel services.

The POSIX 1003.4 Real-Time Extension, however, views the shared memory paradigm as being an important, traditional, high-performance mechanism for interprocess communication in real-time systems. It thus supports shared memory objects as "shared memory special files," i.e., objects named within the standard file system (the use of the file system name space does not imply that management and mapping of shared memory need be implemented using the file management services of the operating system). It enables shared memory special files to be mapped into a process's virtual address space.

Functions

The POSIX 1003.4 shared memory facilities provide the following functions:

- Creating a shared memory special file.
- Opening and closing a specified shared memory special file.
- Mapping (and unmapping) a specified segment of a specified shared memory special file into a process's virtual address space at a specified address.

Notes

- Semaphores (see Synchronization Services discussion) are envisioned as a mechanism for synchronizing access to shared memory.
- The shared memory facilities are designed to be extensible, in that common global objects other than shared memory special files can be mapped into a process's address space through the shared memory facilities, in an implementation-specific way. The only restriction is that the objects be named and accessed through the standard file system facilities. One specific extension singled out in the POSIX 1003.4 Draft 7 document as being particularly relevant in the real-time computing domain is to enable a process to utilize the shared memory facilities to map sections of physical address space into its virtual address space.
- The shared memory facilities are designed to support high- performance data sharing, and, in particular, not shared libraries. "The features of executable shared memory special files and mapping shared memory with execute-only permission are open issues" [POSIX 1003.4 Draft 7, p. 56].

In addition, the POSIX 1003.4 Real-Time Extension working group supports the notion that a process should be able to lock its address space, or specified regions thereof, into memory. Such a capability is viewed as being crucial to deterministic high

performance, which is essential in real-time systems.

Functions

The POSIX 1003.4 process memory locking facilities provide the following functions:

- Locking and unlocking specified regions of a process's address space into memory.

Notes

- The process memory locking interface defined by POSIX 1003.4 enables processes to specify certain "logical" regions of their address space for locking. These include the data region, the text region, the stack region, the shared memory region, and the executable region. However, it recognizes that some systems cannot support locking such memory regions separately; therefore it makes the locking of logical regions optional.

17. Scheduling Services

The POSIX standard (IEEE 1003.1-1988) currently does not define the process scheduling to be performed. The POSIX 1003.4 Real-Time Extension, however, views preemptive, dynamic priority-driven scheduling as being fundamental to real-time systems. It supports two variants of preemptive, dynamic-priority-driven scheduling. The variants are distinguished by the way in which processes of equal priority are scheduled. In the first variant, runnable processes of equal priority are scheduled according to a first-in-first-out (FIFO) policy.(It should be noted that if a process sets its priority to its current priority, the process is viewed as "entering" the queue; so, it becomes the last, or newest, member of the queue, regardless of its previous position.)In the second variant, runnable processes of equal priority are scheduled according to a round-robin (RR) policy, with a specified time slice.

Functions

The POSIX 1003.4 scheduling facilities provide the following functions:

- Setting the priority of and getting the priority of a specified process.
- Setting the "scheduling policy" of and getting the scheduling policy of a specified process. The scheduling policy can be: (1) preemptive, dynamic-priority-driven, FIFO within a priority level, (2) preemptive, dynamic-priority-driven, RR within a priority level, with a specified time slice, or (3) implementation-specific.

Notes

- It is recognized in the POSIX 1003.4 Draft 7 document that issues pertaining to priority inversion, preemption of (non-processor) resources, and lightweight processes remain to be addressed.

18. Synchronization Services

The POSIX 1003.4 Real-Time Extension adopts the binary semaphore as the basic means of process synchronization. It notes that the binary semaphore is a "minimal" synchronization mechanism, and that other mechanisms such as counting semaphores and monitors can be implemented on top of the binary semaphore. It supports semaphores as "semaphore special files," i.e., objects named within the file system (although it must be noted that the standard does not imply that the semaphore functions need be performed by the file management portions of the operating system. In fact, the standard explicitly defines this facility in such a way as to allow implementations to avoid system calls for successful semaphore accesses.).

Functions

The POSIX 1003.4 semaphore facilities provide the following functions:

- Creating a semaphore special file.
- Opening and closing a specified semaphore special file.
- Doing a P-operation (Dijkstra, "Co-operating Sequential Processes", 1968) on a semaphore represented by a specified semaphore special file. The P-operation can be invoked either unconditionally or conditionally. When invoked conditionally, the P-operation is performed only if the semaphore is in an unlocked state, in which case the P-operation causes the semaphore to enter a locked state and the invoking process to become the holder of the semaphore. It should be noted that blocked processes are granted the semaphore in priority order, with FIFO ordering for processes of equal priority.
- Doing a V-operation on a semaphore represented by a specified semaphore special file. The V-operation can be invoked either unconditionally or conditionally. When invoked conditionally, the V-operation is performed only if other processes are currently blocked by the semaphore.

Notes

The POSIX 1003.4 document notes [POSIX 1003.4 Draft 7, p.61]: "Since semaphores have been defined as a special file, they may be opened by two processes that do

not share physical address space if a distributed file system is utilized." It goes on to state [POSIX 1003.4 Draft 7, p. 67]: "Semaphores are only required to operate when the processes using a common semaphore have the same physical address space. If a distributed file system is used, a mechanism shall be provided to ensure that only processes that have the same physical address space can access the semaphore."

19. System Initialization and Reinitialization Services

The POSIX interface specification leaves system initialization and reinitialization services to be defined by the implementation.

20. System Operator Services

The POSIX interface specification leaves system operator services to be defined by the implementation.

21. Time Services

The POSIX 1003.1-1988 standard provides for interrogating and reading time and date, as well as a sleep() function to delay for a set period of time. The standard defines these functions in units which are, however, unacceptably coarse for use by real-time systems.

The POSIX 1003.4 Real-Time Extension to POSIX provides additional fine resolution interfaces to system-wide timers and to per-process interval timers that make time visible to processes and enable processes to schedule timer events in a variety of useful ways. It views such interfaces as being essential to real-time systems, which are distinguished by the significance of the role that time and timing constraints play in them.

Functions

The POSIX 1003.4 timer facilities provide the following functions:

- Setting the value of, getting the value of, and getting the resolution of a specified system-wide timer.
- Creating and destroying a per-process interval timer, based upon a specified system-wide timer and a specified delivery mechanism (signals, events, or implementation-specific). If events are specified, then the application programmer defines an event and writes an event trap routine in accordance with the POSIX 1003.4 asynchronous event notification facilities (see Event and Error Management Services).

- Setting the value of, getting the value of, and getting the resolution of a specified per-process interval timer. The "value" of an interval timer consists of two parts: (1) timer interval and (2) remaining time to the next timer expiration. The remaining time to the next timer expiration can be set to a given offset from the current time (as known by the associated system-wide timer), or to a given absolute value. The timer interval, if nonzero, indicates that periodic timer expirations are to begin occurring after the initial timer expiration, where the period is equal to the specified timer interval.

Notes

- The data structure that is used to represent time provides for nanosecond resolution.

22. Proprietary or Open

As an IEEE standard, POSIX is fully open for implementation by any operating system vendor.

23. Qualification as a Standard

The POSIX standard is controlled by IEEE (P1003 committee).

24. Platform Flexibility

There are no known hardware platforms unable to support a conforming POSIX implementation.

V: Detailed Description

Submitted by Karen Gordon (IDA)

The V distributed operating system, developed under the leadership of David Cheriton at Stanford University, is 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.
- V's target application domains include real-time, interactive timesharing, and batch applications. Real-time requirements have always been a major consideration. Interactive timesharing has been the primary application, though. V is used to transform a cluster of workstations into a distributed system that offers users the same resource and information sharing capabilities traditionally provided by a centralized timesharing system. Recent work has investigated the possibility of supporting large distributed parallel applications.

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.

1. Language Support Services

- Ada language support services.
- Support for other languages. 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.

2. Architecture Dependent Services

- Internetwork communication. V incorporates a system server known as the Internet server, which implements the DoD TCP/IP suite of protocols.

3. Capability and Security Services

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

VMTP, the transport protocol underlying V IPC, incorporates security mechanisms, including "entity domains" and encryption. In VMTP, direct communication can occur only on an intra-domain basis, thus ensuring the isolation between security levels required for mandatory access control. The idea is to have one domain per security level. Entities can belong to more than one domain, so trusted servers could communicate with users of different security levels. Encryption can be used as a mechanism to facilitate the secure authentication of subjects required for discretionary access control.

4. Data Base Services

5. Data Interchange Services

6. Event and Error Management Services

V incorporates an exception server outside the kernel. The kernel process server causes the exception-incurring process to send a message describing its problem to the exception server. The exception server then takes action, for example, invoking an interactive debugger.

7. File Services

- File Server. V implements file services outside the kernel via the file server, which implements a UNIX-like file system.

  The file system utilizes a contiguous allocation scheme that results in most files being data contiguous on the disk.

- Naming. 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).

8. Generalized I/O Services

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.

9. Graphics Services

V incorporates a display server, which supports multiple views, zooming, and redraw.

10. LPOS to LPOS Communication Services

The V kernel is replicated at each participating network node. The kernels cooperate to provide the image of a single unified distributed system, in which processes execute in address spaces and communicate using V IPC. Kernel services (e.g., process management, memory management, communication management, device management) are themselves invoked via V IPC.

11. Man-Machine Interface (MMI) Services

The V display server implements multi-window facilities using a bitmap display.

12. Networks and Communications

- 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 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.

In part to support real-time applications, VMTP provides datagram message transactions, prioritized message transmission and delivery, and conditional message delivery (i.e., delivery only if the receiver is awaiting a message when the message arrives).

- Pipe Server. V incorporates a pipe server that implements UNIX-like pipes.

13. Process Management Services

- 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.

- Kernel process server. The kernel process server implements operations to create, destroy, query, modify, and migrate processes.

14. Project Support Environment Services

15. Reliability, Adaptability, and Maintainability Services

- Reliability and fault tolerance. V supports the notion of a process group as a set of processes identified by a "group identifier." The processes may reside at any node in the distributed system. The process group mechanism and multicast communication are used to implement distributed and replicated services. Both distribution and replication enhance reliability and fault tolerance.

16. Resource Management Services

- 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.

  In part to support real-time applications, V enables programs to be specified as memory-resident.

- Device Management. The kernel device server implements access to devices supported by the kernel, including disk, network interface, mouse, frame buffer, keyboard, serial line, and tape. The device server is device-independent code that interfaces between the process-level client and the driver modules for the individual devices. The device server implements the UIO interface.

  Process-level servers (e.g., file server, Internet server, display server) implement extended abstractions using the basic interfaces provided by the kernel device server.

17. Scheduling Services

In regard to processor scheduling, the kernel provides simple priority-based scheduling. In the uniprocessor case, the kernel allocates the processor to the highest priority process in the ready queue. In the multiprocessor case, a process is associated with a processor and its ready queue. The kernel schedules processes so that each processor is always executing the highest priority process in its own ready queue. In addition, the kernel periodically attempts to balance the load by changing the process-to-processor associations.

Above the kernel, a d dicated scheduler process implements a higher level of scheduling. The scheduler manipulates priorities to effect time-slicing among interactive and background processes. A number of high priority levels are reserved for real-time processes and are not subject to the priority manipulations of the scheduler.

18. Synchronization Services

19. System Initialization and Reinitialization Services

20. System Operator Services

21. Time Services

One of the V kernel servers is a time server. The kernel time server maintains the current time of day, and it enables a process to read the time, set the time, and delay for a specified period of time. An operation for awaking a process that is delaying is also provided.

Time synchronization across nodes is implemented by a process outside the kernel.

22. Proprietary or Open

The protocols and interfaces are open.

23. Qualification as a Standard

The V project emphasizes protocols and interfaces as a means of defining and building distributed systems. Efforts are underway to promulgate some of its protocols, most notably VMTP, through the DoD data communication protocol standards process. The naming and I/O protocols represent significant contributions to distributed system technology, and have played major roles in the development of the V distributed operating system. Other protocols of interest include ones for remote execution, migration, time synchronization, and atomic transactions.

24. Platform Flexibility

V is a distributed operating system designed for a cluster of workstations interconnected by a high-performance network. It currently runs on SUN and MicroVAX workstations, which are interconnected by a 10-megabit Ethernet.

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.

25. References

Primary

[Cheriton 88b] Cheriton, David R., "The V Distributed System," *Communications of the ACM 31*, 3 (March 1988), 314-333. This article presents an excellent overview of the V distributed operating system. It is the primary source for the above summary. Some of the material in the summary was taken verbatim from the article.

Secondary

[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 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.

Real-Time Operating System Technology[1]
Submitted by Karen Gordon (IDA)

1. REAL-TIME COMPUTING SYSTEMS

Real-time computing systems are designated as such because of the significance of the role that the time dimension plays in them. In the premier issue (dated June 1989) of *The Journal of Real-Time Systems*, the introductory editorial characterizes real-time systems as that category of systems in which "the correctness of the system depends not only on the logical results of computations but also on the time at which the results are produced" [Stankovic 89, p. 6].

Thus, in real-time computing systems, timeliness is mandatory. Timing constraints are imposed by the environment in which the real-time computing system exists. Typically, the environment consists of a larger *controlled* system (e.g., automobile, aircraft, ship, submarine, missile, hospital patient monitoring system, air traffic control system, factory floor, nuclear power plant, etc.), which is in turn embedded in and affected by its physical environment. The real-time computing system is the *controlling* system. Failure to meet environment-imposed timing constraints can have catastrophic consequences, such as loss of life, loss of the controlled system, or failure of the mission of the controlled system.

The qualifiers "hard" and "soft" are often used in conjunction with the term "real-time system." While precise definitions have not been agreed upon, the general distinction seems to lie in the nature of the timing constraints. *Hard real-time systems* have timing constraints that are both rigid and mandatory. For example, a task may have an absolute upper bound on response time that must never be exceeded. Such an upper bound is referred to as a hard deadline. *Soft real-time systems*, on the other hand, have more flexible timing constraints (sometimes referred to as soft deadlines). In the example, flexibility could entail (1) relaxing the requirement that the upper bound *never* be exceeded, by moving from deterministic performance specifications to stochastic specifications (e.g., response to an operator action must occur within 350 milliseconds with 97% probability), or (2) relaxing the upper bound itself, so that a response computed after the "upper bound" is still usable, although in some sense less valuable [Jensen 85] [Locke 86], or (3) some combination of the above.

---

1. This paper was included as Section 3.4 of Version 0.7 (dated 01/18/1990) of the *NGCR OSSWG Available Technology Report*. However, due to an OSSWG management decision to streamline the report by removing discussions of mission-critical operating system issues, this paper and other issue papers were not included in Version 1.3 (dated 09/14/1990) of the *NGCR OSSWG Available Technology Report*, which was published in the OSSWG's *First Annual Report*.

2. REAL-TIME APPLICATION DEVELOPMENT FRAMEWORKS

In general, development of real-time applications is undertaken in one of two divergent frameworks: a periodic framework or an aperiodic, asynchronous event-driven framework. In a periodic framework, an application is developed as a collection of periodic tasks. A periodic task is one that is initiated at regular time intervals, or *periods*.

The periodic framework has the advantage that timing constraints are explicitly taken into account through the period mechanism. The end of each period is the hard deadline for the task initiated at the beginning of the period. Period lengths are chosen so that an application can "keep pace" with its environment. Since task arrivals are synchronous, a system can be sized to guarantee that all deadlines are met. The disadvantage of the *purely* periodic framework is that it does not accommodate asynchronous events (or aperiodic tasks), which inevitably occur in all but the simplest control systems.

However, the periodic framework can be adapted to deal with asynchronous events in various ways. For example, a periodic task can be created to service asynchronous events (in which case the periodic task in effect "polls" for event occurrences), or asynchronous events can be processed as *background* tasks. In the June 1989 issue of *The Journal of Real-Time Systems* [Sprunt 89], Sprunt, Sha, and Lehoczky present methods for scheduling aperiodic tasks in a specific periodic framework—the rate monotonic framework.[2] Their methods provide lower average response times for aperiodic tasks than either polling or background processing, while at the same time maintaining guarantees of meeting all periodic task deadlines.

In an aperiodic framework, processing is not primarily periodic but is instead event-driven. Events occur asynchronously. Typically, priorities are used to establish a service ordering for events. In general, the resource management objective is to process events as fast as possible, subject to their priorities. In this type of framework, it is periodic tasks that are anomalous. The problem is not the fact that their interarrival times are constant but the fact that they have *hard deadlines* and expect *100%* to be met. Unless sufficient processor time is *reserved* for periodic tasks, for example, by giving all periodic tasks higher priorities than all aperiodic tasks, then providing such deterministic assurance is infeasible. Aperiodic tasks with hard deadlines suffer from the same lack of deterministic assurance, unless they are *well-behaved* in the sense of having some reasonable minimum separation time and can have sufficient processor time dedicated to them. In general, as soon as a stochastic component (namely, aperiodic tasks or asynchronous

2. In a rate monotonic framework, tasks are assigned static priorities according to their arrival rates: tasks with higher rates are assigned higher priorities. Task scheduling is preemptive priority-driven. The rate monotonic algorithm was shown to be *optimal* among the class of preemptive, static priority driven algorithm by Liu and Layland [Liu and Layland 73].

events) is introduced into a system's workload, assurance must be cast in stochastic terms rather than absolute, deterministic terms.

The disadvantage of the traditional (priority-driven) aperiodic framework is that it lacks mechanisms for dealing with time. Timing constraints are not specified, either explicitly or implicitly. Consequently, methods are not available for ensuring that the timing constraints are met. The application developer is afforded little support in *sizing* the system to ensure that "as fast as possible" is indeed fast enough. In efforts to address this concern, methods for introducing timing constraints into aperiodic frameworks have been proposed and investigated [Jensen 85] [Locke 86], but have not yet made the transition to become a part of the state of the practice in real-time computing.

3. OPERATING SYSTEM SUPPORT FOR REAL-TIME COMPUTING

As recently pointed out by Stankovic [Stankovic 88], today's real-time systems are built through brute-force techniques and at inordinate expense. As systems become ever larger and more complex, a more "scientific" approach is called for. An important ingredient of any scientific approach is an operating system designed to meet the unique needs of real-time computing, i.e., a *real-time operating system*.

Operating systems that claim to be real-time generally offer one or more of the following categories of services and features: mission-driven/application-directed resource management, timely response to events, predictable service times and overhead times, and time services that make time visible and accessible to applications. The significance of each of these categories is discussed below. It should be noted that most real-time operating systems are targeted to only one of the two real-time application development frameworks described above and that the importance of the different categories of services and features varies according to the target framework. For example, time services, specifically ones that enable periodic initiation of tasks, are more critical in a periodic development framework, while timely response to events is more critical in an aperiodic framework.

3.1 MISSION-DRIVEN/APPLICATION-DIRECTED RESOURCE MANAGEMENT

Over the past two decades, operating system research has been focused primarily on interactive computing. Common resource management goals have been to minimize average delay, maximize average throughput, and ensure "fairness" to competing users. While such efficiency-related and fairness-related goals may be well suited to the requirements of interactive computing, they do not adequately meet the requirements of

real-time computing. As discussed above, real-time computing systems are distinguished by the presence of environment-imposed timing constraints.

A real-time operating system must be designed in accordance with the fact that a real-time computing system exists to perform a mission. The operating system should be supportive of the mission: the resource management provided by the operating system should be neither efficiency-driven nor fairness-driven, but *mission-driven*. In particular, resource management should be driven by the time constraints of the mission, as conveyed to the operating system by the application. It is the responsibility of the application to specify resource management attributes to the operating system, and it is the responsibility of the operating system to manage *all* resources according to the application-specified attributes.

Mission-driven, application-directed resource management entails the following:

a. Application-directed allocation: In order to provide fast and predictable performance, a real-time operating system should enable an application programmer to specify certain "allocation attributes." For example, an application programmer might need to specify that a given program be memory-resident or that a given file be contiguous.

b. Application-directed scheduling: Applications should have the capability to specify certain "scheduling attributes" that enable the operating system to impose an effective ordering on tasks or events. The operating system should take the scheduling attributes into account whenever contention or queueing occurs.

   The issue of exactly what the scheduling attributes should be is the topic of considerable controversy in the research community. Some believe that preemptive priorities are sufficient; others contend that the concept of priority must be broken down into complementary aspects of "urgency" and "importance," where urgency is meant to capture nearness of deadlines and "importance" is meant to capture criticality to the mission [Jensen 85]. A few go further, introducing even more resource management attributes, and making the scheduling problem even more complex.

c. Application-directed synchronization: The synchronization of concurrent activities should be controllable by an application. For example, an application should be able to specify whether operations (e.g., I/O, message passing) are to be done synchronously or asynchronously. While asynchronous operations introduce some complexity, they have been found to be useful in

real-time applications. The reasoning behind the provision of asynchronous operations is that synchronous operations may *unnecessarily* impede the forward progress of a path of execution.

3.2 TIMELY RESPONSE TO EVENTS

A real-time system must maintain its integrity with respect to the state of its environment. This can be viewed as a requirement to maintain *external* consistency, i.e., consistency between the actual state of the environment and the real-time system's perceived state of the environment. At the same time, *internal* consistency must also be maintained. That is, multiple concurrent tasks that constitute an application must have accurate perceptions of the states of one another.

If occurrences that alter the state of the environment or the system itself are viewed as *events*, then what is required is timely response to events. In other words, a real-time operating system should be able to respond to both external and internal events in a timely—both fast and predictable—manner; moreover, it should ensure that applications can also respond to events in a timely manner, through timely event notifications to applications.

3.3 PREDICTABLE SERVICE TIMES AND OVERHEAD TIMES

To facilitate the predictability of the performance of a real-time application, the execution times of operating system functions that the application explicitly invokes (via system calls), as well as those that it implicitly invokes, should be bounded. The bounds should not greatly exceed the means; otherwise, excessive resources may have to be dedicated to the application to assure acceptable performance.

3.4 TIME SERVICES

A real-time operating system should provide services that make time visible and accessible to applications. For example, applications should be able to set the time, read the time, and schedule events to occur at specified times, such as at periodic time intervals.

4. EXISTING REAL-TIME OPERATING SYSTEMS

4.1 TABLE-DRIVEN REAL-TIME EXECUTIVES: CYCLIC EXECUTIVES

Cyclic executives are designed to support a periodic application development framework. They are discussed in depth by Baker and Shaw in the June 1989 issue of *The Journal of Real-Time Systems* [Baker and Shaw 89]. Briefly speaking, a cyclic executive has a single responsibility: to interleave the executions of periodic tasks according to a

fixed, predetermined schedule. The schedule is often specified in a static scheduling table, which indicates which pieces of which tasks are to be executed in what order during each time frame of each scheduling cycle. The scheduling table is formulated by the application developer as a key part of the application development. The formulation of a scheduling table can be viewed as a bin-packing problem, in which tasks are fit into cycles in such a way that all deadlines can be met.

The real-time services and features described above are addressed by cyclic executives as follows:

a. Mission-driven/application-directed resource management: Resource management decisions are deterministic. They are dictated by the application developer via the scheduling table.

b. Timely response to events: Asynchronous events and aperiodic tasks typically are handled by mechanisms such as polling or background processing. Timely response can be achieved through *frequent* polling.

c. Predictability of service times and overhead times: A cyclic executive is essentially a table-driven scheduler. Predictability of service times is a non-issue, since services (other than scheduling) are not offered. Predictability of overhead can be accomplished by straightforward measurement, since the only function of the cyclic executive—scheduling—is driven by static scheduling tables, which capture scheduling decisions made off-line by the application developer.

d. Time services: Periodic timer interrupts are vital to cyclic executives. They dictate frame boundaries and synchronize the system to the scheduling table.

4.2 PRIORITY-DRIVEN REAL-TIME EXECUTIVES

Stankovic and Ramamritham offer a concise description of today's priority-driven real-time executives in a paper presented at the 1987 Real-Time Systems Symposium [Stankovic and Ramamritham 87]. In the paper, they characterize most existing real-time executives or kernels as being "stripped down and optimized versions of timesharing operating systems." This is not surprising, given the previously noted fact that operating system research and development over the past two decades has been focused on the interactive or timesharing domain. In effect, familiar operating system concepts have been and are continuing to be used to construct new operating systems that can meet the unique demands of real-time computing. The two major design objectives are (1) minimization of overhead, which is the motivation behind the effort to "strip down" general-purpose timesharing operating systems, and (2) speed, which is the

motivation behind optimization efforts.

Stankovic and Ramamritham go on to enumerate several specific characteristics of today's priority-driven real-time executives [Stankovic and Ramamritham, p.146]. Below, we quote those characteristics in terms of our four general categories of real-time services and features:

a. Mission-driven/application-directed resource management: (1) priority scheduling, (2) the ability to lock code and data in memory,[3] and (3) the presence of special sequential files that can accumulate data at a fast rate.

b. Timely response to events: (1) the ability to respond to external interrupts quickly, (2) the minimization of intervals during which interrupts are disabled, (3) multi-tasking with task coordination being supported by features such as mailboxes, events, signals, and semaphores.

c. Predictability of service times and overhead times: (1) a small size (with its associated minimal functionality), (2) fixed or variable sized partitions for memory management (no virtual memory), and (3) a fast context switch.

d. Time services: (1) support of a real-time clock, (2) primitives to delay tasks for a fixed amount of time and to pause/resume tasks, and (3) special alarms and timeouts.

It should be noted that placement of the characteristics under the four service/feature categories involved some judgment calls, because some of the characteristics play roles in more than one category. For example, fast context switching contributes to timely response to events, as well as to predictability.

Priority-driven real-time executives as described above can support aperiodic real-time application development frameworks, as well as priority-driven periodic frameworks such as the rate monotonic framework.

4.3 PRIORITY-DRIVEN REAL-TIME OPERATING SYSTEMS

In this section, we discuss a class of real-time operating system that is closely related to the above class. The distinction between the two is that the real-time *executives* discussed above are intended solely for real-time computing, whereas the real-time *operating systems* of this section are, in a sense, general-purpose operating systems capable of meeting the demands of real-time computing.

---

3. This characteristic is quoted from a revised list that appears in the IEEE Computer Society Press Tutorial *Hard Real-Time Systems* [Stankovic and Ramamritham 88, p. 4].

Rather than speaking in generalities, we focus on the IEEE P1003.4 RealTime Extension for Portable Operating Systems (POSIX 1003.4). It is useful to do so, because POSIX 1003.4 arguably captures the state-of-the-practice in real-time operating systems.

Since POSIX 1003.4 is described in depth later in this document (in Section 5.4.4 of Version 0.7 of the *NGCR OSSWG Available Technology Report*), we simply put it into context here, by presenting its interfaces in terms of our four real-time service/feature categories:

a. Mission-driven/application-directed resource management: In this vein, POSIX 1003.4 offers (1) preemptive, dynamic priority-driven scheduling, (2) process memory locking, (3) real-time files, (4) asynchronous I/O, and (5) synchronized I/O. Additionally, POSIX 1003.4 attempts to take priorities into account in every interface in which contention or queueing can occur.

b. Timely response to events: To support this, POSIX 1003.4 offers (1) interprocess communication in the form of shared memory and semaphores, as well as in the form of message passing, (2) asynchronous event notification, and (3) threads, or lightweight processes.

c. Predictability of service times and overhead times: The POSIX 1003.4 philosophy here is to define metrics for each of its interfaces and to require vendors to report the values of the metrics. Thus, standard *metrics* are defined, but standard *values* are not.

d. Time services: POSIX 1003.4 provides interfaces to system-wide timers and per-process interval timers that make time visible to processes and enable processes to schedule timer events in a variety of useful ways, including periodically.

POSIX 1003.4, as well as priority-driven real-time operating systems in general, can support aperiodic real-time application development frameworks, as well as priority-driven periodic frameworks such as the rate monotonic framework.

REFERENCES

[Baker and Shaw 89] Baker, T.P. and A. Shaw, "The Cyclic Executive Model and Ada," *The Journal of Real-Time Systems 1*, 1 (June 1989), 7-25.

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

[Liu and Layland 73] Liu, C.L. and J.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.

[Sprunt 89] Sprunt, B., L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems," *The Journal of Real-Time Systems 1*, 1 (June 1989), 27-60.

[Stankovic 88] Stankovic, J.A., "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems," *IEEE Computer 21*, 10 (October 1988), 10-19.

[Stankovic 89] Stankovic, J.A., W.A. Halang, and M. Tokoro, editors, "Editorial," *The Journal of Real-Time Systems 1*, 1 (June 1989), 5-6.

[Stankovic and Ramamritham 87] Stankovic, J.A. and K. Ramamritham, "The Design of the Spring Kernel," *Proceedings of the Real-Time Systems Symposium*, December 1987, 146-157.

[Stankovic and Ramamritham 88] Stankovic, J.A. and K. Ramamritham, *Hard Real-Time Systems*, Computer Society Press of the IEEE, Washington, D.C., 1988.

PART 3

Contributions to the

NGCR OSSWG Recommendation Report[1]

In this part is a copy of the IDA contribution to the NGCR OSSWG Recommendation Report. The IDA contribution provides rationale for the choice of a single candidate as the NGCR operating system interface standard baseline.

1. This report was published as the *Recommendation Report for the Next-Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline*, D.P. Juttelstad (editor), Technical Document 6902, Naval Underwater Systems Center, Newport, Rhode Island, 1 June 1990.

Rationale for Single-Candidate Baseline
Submitted by Karen Gordon (IDA)

The charter of the NGCR OSSWG is to establish a commercially-based family of operating system interface standards for use in the development and deployment of Navy mission-critical computing systems in the mid-1990s and beyond. Candidates for NGCR standardization include existing public interface standards, as well as existing interface definitions (for example, based on commercial products or research prototypes) that could become public standards. The goal of the OSSWG has been to seek to adopt existing standard/definition(s), if possible, and to resort to *Navy adaptations* of existing standards/definitions, or (in the worst case) *Navy-created* standards only if demanded by technical considerations.

Almost from the outset, as evident in the charter, the OSSWG anticipated that technical considerations would prevent the Navy from adopting a single existing operating system interface standard/definition as the NGCR standard. The OSSWG believed that, at the very least, the NGCR standard would have to comprise multiple existing standards/definitions. The OSSWG was prepared to find a composite solution buried in the Evaluation Results. The Evaluation Process was developed with this mind-set; it was designed not to compute a single winner, but instead to reveal multiple winners, which together would constitute the composite solution.

It was envisioned that multiple winners would be revealed through two mechanisms: (1) separation of technical issues into 16 service classes, with scores computed for each service class on the basis of Weight Set 1, and (2) formulation of representation application domains (RADs), with scores computed for each RAD on the basis of Weight Set 2. The expectation was that each candidate would be better suited to some service classes and to some RADs than to other service classes and RADs. The RADs, in particular, were intended to reveal different "winners" for different domains.

Needless to say, the OSSWG was surprised when the RAD results were computed. The RADs failed to point out strengths and weaknesses of the candidates. For each candidate, scores varied hardly at all across the seemingly disparate application domains. According to the RAD results, a candidate's effectiveness simply was not a function of application domain.

The OSSWG struggled with the RAD results during the March 1990 meeting (and for days before and weeks after). The OSSWG questioned the formulation of the RADs. Numerous variations on the RADs were defined, and the raw evaluation data was applied to the variations. The results did not change appreciably. At the March 1990 meeting, the Available Technology Subgroup constructed two new sets of application

domains (in an ad hoc manner) and then considered the candidates in light of the domains. Almost every candidate was "suitable" for every domain. The major exception was Cronus, which was judged by the Available Technology Subgroup to be unsuitable for real-time applications.

The OSSWG considered developing a composite solution from another viewpoint–that of layering, with the layers being kernel, LPOS, and SRAX. At the March 1990 meeting, the Available Technology Subgroup considered the utility of defining a composite standard in terms of these layers. It concluded that such a composite standard was not called for. One of the arguments against adopting per-layer standards was that only the LPOS layer is ready for standardization. The LPOS layer is well understood and well represented in the OSSWG Requirements Document. Perhaps most importantly, there seems to be substantial support for a particular interface at the LPOS layer (namely, POSIX). The kernel layer and SRAX layer have not reached the same levels of understanding or consensus.

The Available Technology Subgroup considered what it would mean to the NGCR Program to (initially) limit standardization to the LPOS layer. It concluded that a standard that addresses only the LPOS layer indeed would be useful and not handicapping. The goal is application portability, not LPOS portability. Any kernel-like functionality that is demanded by applications could be (and already is in some candidates) incorporated into the LPOS layer. That is, specific kernel-like services could be reached through the LPOS without (implicitly) embedding a complete kernel in the NGCR standard. Moreover, by not mandating a particular kernel, the Navy would gain the advantage that it could have its LPOS interface standard implemented on many different kernels (maybe different ones for different applications). In other words, the Navy could obtain the necessary functionality without unnecessarily constraining implementations of the NGCR operating system interface standard. As for the SRAX layer, its functionality could be supplied in part by network protocols and in part by applications themselves, as it is today. While further standardization at the SRAX may be desirable in the long term, it is probably not desirable at this time.

In short, the OSSWG realized that the Evaluation Results did not point to a composite solution; per-RAD winners did not come forward, and the concept of per-layer winners was rejected. The results of the technical evaluation indicated that three candidates might be acceptable. The OSSWG decided that the three candidates should be carefully examined to validate the Evaluation Results and to aid in interpretation of the Evaluation Results. The scores on programmatic issues should be taken into account in this examination. Then, if two or three candidates proved to be acceptable from a technical standpoint *and* from a programmatic standpoint, one of the acceptable candidates

should be singled out for recommendation by the OSSWG to the NGCR Program Office as the basis for NGCR operating system interface standardization. The OSSWG believes that, under the circumstances, a multiple-candidate solution would be disadvantageous; a multiple-candidate solution would (1) dilute NGCR resources and (2) fracture industry support. To maximize Navy influence and to achieve cost effectiveness, the NGCR Program should focus its efforts on a single candidate.

It should be noted that by recommending a single-candidate solution, the OSSWG is not ruling out the possibility of a family of standards. All that is ruled out is the possibility of a family based on an NGCR collection of divergent candidates. A family based on a single candidate is still possible and in fact probable. At this point, the OSSWG believes that the "family" will take the form of a "series" or "set" of NGCR Interface Standards that can be tailored or scaled to any particular application (interface) requirements. It is anticipated that defining the precise form of tailoring will be a major order of business for the next phase of the OSSWG.

PART 4

Contributions to the

NGCR OSSWG After-Action Report[1]

In this part are copies of the following IDA contributions to the NGCR OSSWG After-Action Report:

- Utility of the Evaluation Process Itself
- Credibility of the Raw Scores

1. This report was published as the *After-Action Report for the Next-Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline Selection Process*, J.T. Oblinger (editor), Technical Document 6904, Naval Underwater Systems Center, Newport, Rhode Island, 1 June 1990.

Utility of the Evaluation Process Itself
Submitted by Karen Gordon (IDA)

The evaluation process itself (as opposed to its results) played an invaluable role in the OSSWG efforts: the evaluation process was the catalyst that enabled consensus building to occur.

First, by forcing evaluators to walk through documentation on each candidate and to consider each candidate in terms of many specific requirements, the scoring phase of the evaluation process increased the evaluators' objective knowledge of the candidates. Moreover, and of no less consequence, it helped evaluators to develop stronger subjective or intuitive feelings about the candidates.

Then, in the analysis phase, the evaluation process drove the OSSWG to consider the candidates from another viewpoint. "Why did the RAD scores turn out so uniform for each candidate?" It was expected that any given candidate would score much higher in some RADs than in others. "What is it about the RADs or about the candidates that makes a candidate's effectiveness indistinguishable with respect to such seemingly divergent RADs?" The struggle with these questions shed new light on the candidates, as well as on the overall task of the OSSWG.

Credibility of the Raw Scores

Submitted by Karen Gordon (IDA) and Tim Saponas (Intel)

When confronted with the results of preliminary analysis, the OSSWG faced the question of "Do the numbers really mean anything?" Three key points caused concern:

1. The dispersion of the raw scores (for a given criterion for a given candidate) was much larger than anticipated or desired. For example, certain candidates had extremely large sigmas which often were a result of an extremely widely spread bimodal distribution.

2. In some cases, the raw scores for a candidate clearly failed to accurately reflect its capabilities. For example, the results provided a high mean score for a requirement that clearly should be satisfied by a single interface, but further inspection of the candidate's documentation failed to reveal an interface satisfying the requirement.

3. The Representative Application Domain (RAD) scores were uniform across RADs (i.e., for any given candidate, its scores varied little from one RAD to another RAD). See [Section 3.6 of the NGCR OSSWG After-Action Report] for a discussion of the RAD scores.

It was realized that many factors led to these problems, e.g., imprecisely defined candidates, misleading cross matrices, unclear requirements, subjective requirements (such as the programmatic issues and many of the general requirements), varying backgrounds of evaluators, and insufficient opportunity to seek an "all in one room" consensus on the evaluation scores.

The following recommendations possibly would have addressed some of these issues:

1. Screening of candidate documentation. Many of the candidates provided a large amount of documentation much of which had nothing to do with OS interfaces. In some cases, the documentation did not deal with the actual candidate, but rather systems that ran on top of the candidate. A subcommittee of the OSSWG could have been formed to work with the candidate sponsors to ensure that only proper documentation was passed on to the evaluators.

2. Screening of cross matrices. All of the candidate sponsors eventually provided the evaluators with cross matrices that attempted to show how the candidate satisfied each of the requirements. The OSSWG recommended that the sponsors provide the cross matrices but did not specify a format for the document. The result was a set of documents with a wide variety of formats ranging from documents that either

explicitly named interfaces satisfying the requirements (or cited explicit pages in the candidate documentation where the interface could be found) to documents that tried to have general discussions about the requirement with suggestions of possible solutions. The latter tended to mislead the evaluators and avoid the issue of clearly identifying the interfaces. This problem could have been addressed by explicitly requiring a cross matrix from each sponsor that followed a fixed format that required the candidate sponsor to clearly identify the interface satisfying each requirement.

3. Evaluator education on the requirements. While considerable effort went into the refinement of the requirements and the requirements document, the evaluators were never formally briefed on all the requirements. It should be noted that at the December meeting of the OSSWG, the evaluators were provided some clarification on the requirements but only as part of a forum intended for assigning weights. For this case though the group was divided in half with each subgroup dealing with only half the interface. This resulted in several different interpretations of the requirements by the evaluators. A briefing would have allowed evaluators to clarify this confusion and have moved the group towards a more standard interpretation. In addition, it would also have identified the strengths and weaknesses in the requirements document.

4. Lack of a consensus building meeting. The evaluators were asked to perform their evaluation in isolation. They were not brought together until after their evaluations were committed. This left no opportunity for the evaluators to share the knowledge they had obtained and adjust their scores. A consensus building meeting would have allowed this information exchange and moved the OSSWG towards a consensus.

PART 5

Presentation at the

Seventh IEEE Workshop on Real-Time Operating Systems and Software

This workshop was held on May 10-11, 1990, at the University of Virginia in Charlottesville, Virginia. On the first morning of the workshop, a panel session entitled "RT OS Standards: *They are here, but are they good?*" was held. The panel was chaired by Hide Tokuda of Carnegie Mellon University. He invited Karen Gordon of IDA to serve on the panel as a representative of the NGCR OSSWG. In this part is a copy of the presentation given by Karen Gordon as her position statement at the panel session.

OPERATING SYSTEM STANDARDS: THE NGCR OSSWG PERSPECTIVE

Karen Gordon
Institute for Defense Analyses
Member, NGCR OSSWG

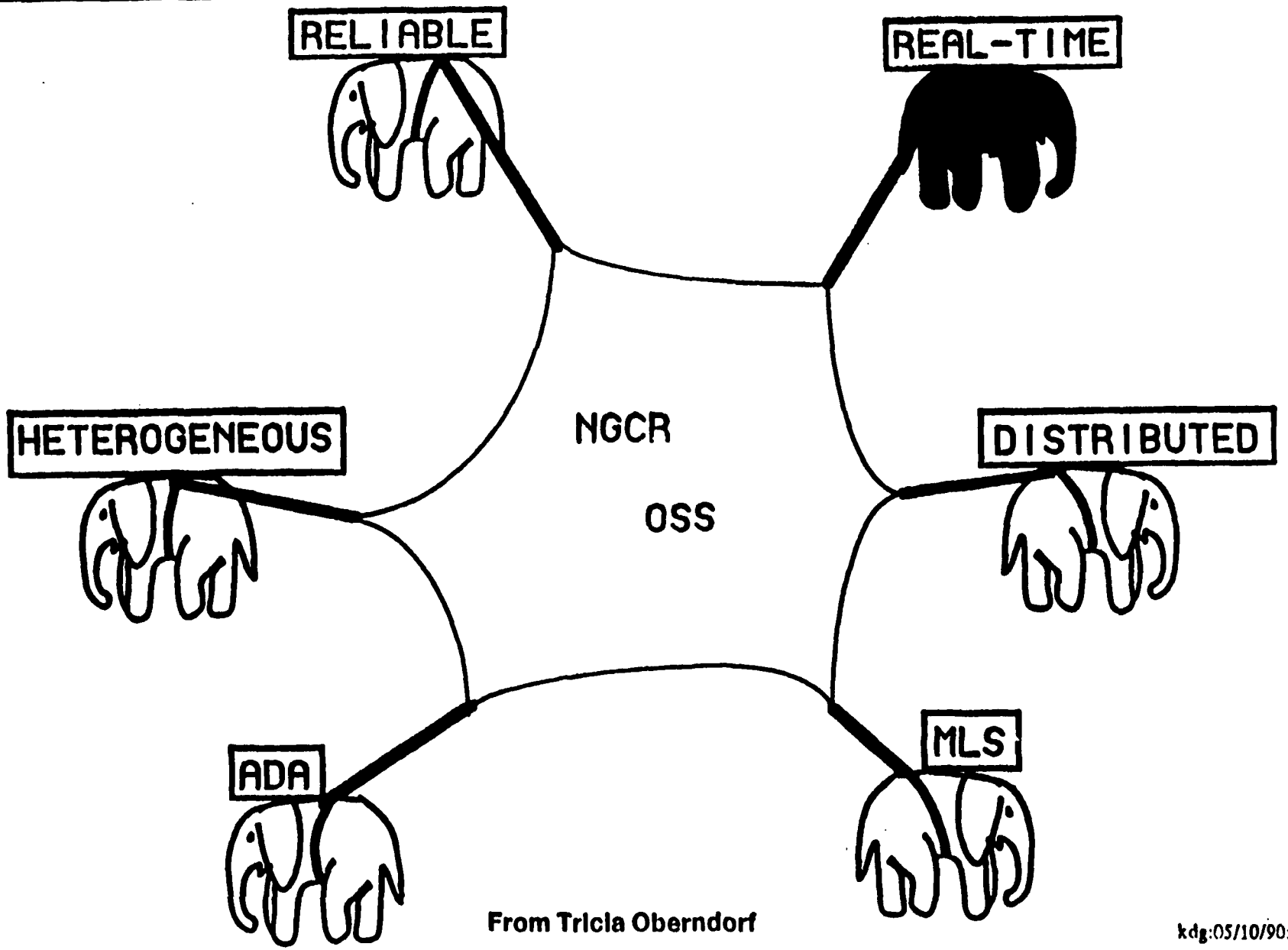
NGCR OSSWG

IDA

- NGCR (Next Generation Computer Resources): a Navy program to revolutionize the Navy's acquisition and use of computer resources
- OSSWG (Operating Systems Standards Working Group): the third in a series of 10 or more NGCR joint Navy/industry/academia working groups
- NGCR OSSWG Charter: "... to establish a commercially-based family of operating systems interface standards for use in the development and deployment of Navy MCCR applications systems in the mid-1990s and beyond..."

NGCR OSSWG DESIDERATA

IDA



From Tricia Oberndorf

NGCR OSSWG ACCOMPLISHMENTS TO DATE

- Mar-Dec 1989: Enumerated requirements
- Mar-Dec 1989: Reviewed operating system standards, implementations, and technology; converged on seven operating system interface standards/definitions as semifinalists
  - — ALPHA, ARTX, CRONUS/SDOS, iRMX, MACH, ORKID, POSIX
- Jan-Mar 1990: Formally evaluated the seven semifinalists with respect to the requirements; converged on three as finalists
  - — ALPHA, iRMX, POSIX
- Apr 1990: Reached consensus; in particular, selected ONE of the three finalists as the foundation on which to build a baseline standard
  - — And the winner is ... POSIX !!!

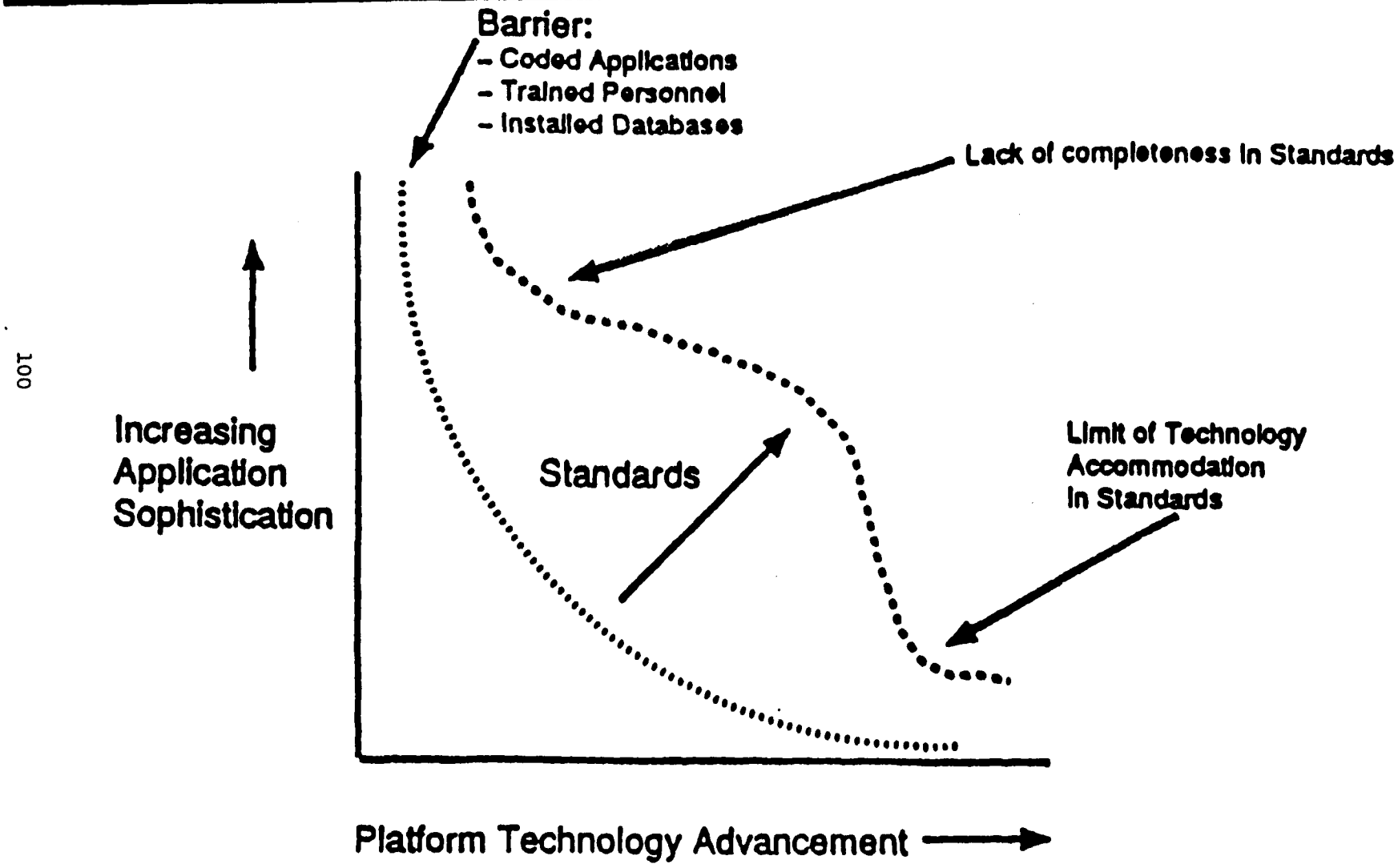
ANTICIPATED BENEFITS OF OPEN SYSTEM STANDARDS

IDA

- COTS (Commercial Off-The-Shelf)
  - Schedule
  - Cost
  - Performance

- Open System Standards
  - Connectivity/Interoperability
  - Portability of programs, data, people
  - Protection of investment in application software

OPEN SYSTEMS POTENTIAL



From "Generic Open System Presentation," Jim Isaak and Kate Comiskey

PART 6

Presentation at the

Joint
Eighth IEEE Workshop on Real-Time Operating Systems and Software
IFAC/IFIP Workshop on Real-Time Programming

This workshop was held on May 15-17, 1991, in Atlanta, Georgia. On the first afternoon of the workshop, a panel session entitled "Operating Systems: Interfaces/Standards" was held. The panel was chaired by Doug Locke of IBM. He invited Karen Gordon of IDA to serve on the panel as a representative of the NGCR OSSWG. In this part is a copy of the presentation given by Karen Gordon as her position statement at the panel session.

POSIX:
THE NAVY NGCR OSSWG PERSPECTIVE

Karen Gordon
Institute for Defense Analyses

NGCR OSSWG

IDA

- NGCR OSSWG: Next Generation Computer Resources Operating Systems Standards Working Group

- Charter: "... to establish a commercially-based family of operating systems interface standards for use in the development and deployment of Navy MCCR applications systems in the mid-1990s and beyond..."

- Phase 1:
  - Mar-Dec 1989: Enumerated requirements and reviewed operating system standards, implementations, and technology
  - Jan-Mar 1990: Formally evaluated seven candidates with respect to the requirements
  - Apr 1990: Selected POSIX as the baseline

- Phase 2 (Apr 1990 - present): Working with POSIX

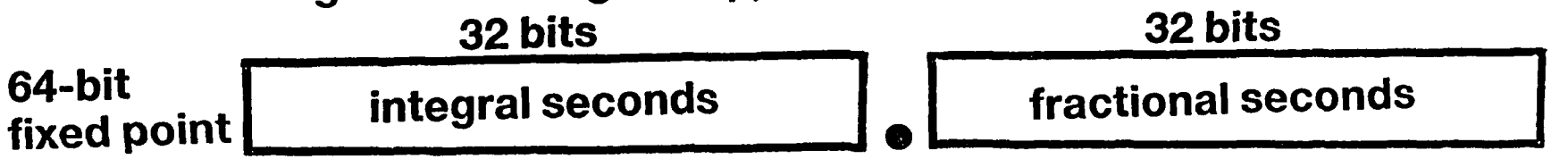
NAVY-PERCEIVED SHORTCOMINGS OF POSIX

IDA

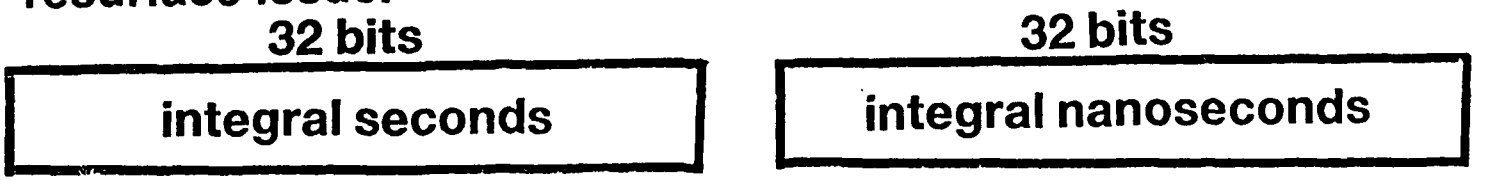
- Timeout mechanism for calls that can block
  - Navy applications need timeout mechanism for all such calls (for fault tolerance)
  - POSIX currently offers (inconsistent) timeout mechanism for only a few such calls. P1003.4 is now addressing.

105

- Time format
  - Navy prefers compatibility with IEEE 1212 (Control and Status Register Working Group) and Network Time Protocol (NTP)



  - P1003.4 considered and rejected IEEE 1212 format. Navy intends to resurface issue.



NAVY-PERCEIVED SHORTCOMINGS OF POSIX (continued)

- Hardware interrupt control and management
  - Navy applications need interfaces for enabling/disabling interrupts, masking/unmasking interrupts, linking interrupt handling routine to interrupt
  - P1003.4 is now addressing

- Memory allocation
  - Navy sees the utility of the concept of "colored" memory (i.e., memory with distinguishing characteristics), which is advocated by VITA and its Open Real-Time Kernel Interface Definition (ORKID).
  - P1003.4 is now addressing

PART 7

Contributions to the

NGCR OSSWG Delta Document[1]

In this part are copies of the following IDA contributions to the NGCR OSSWG Delta Document:

- Data Sheets for Service Class 9 (November 1990)
- Revised Data Sheets for Service Class 9, Criteria 11 and 12 (September 1991)
- Resolution of Deltas for Service Class 9, Criteria 11, 12, and 13 (September 1991)
- Resolution of Deltas for Service Class 13, Criteria 7 and 11 (September 1991)

1. This document currently exists as a working draft entitled *DELTA Document for the Next Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline*.

Data Sheets for Service Class 9 (November 1990)

To be included in Appendix A of the DELTA Document for the Next Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline

Submitted by Karen Gordon (IDA)

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 1
   Criteria Name: Create Process
   Requirement Document Paragraph: 2.9.1

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 49 (fork) and 50 (exec)
   Paragraph: 3.1.1 (fork) and 3.1.2 (exec)

   Capabilities Summary: Creating (and starting) a new process is accomplished via a sequence of two function calls, (1) fork and (2) one of the functions in the exec family. The fork function creates a new (child) process, which is a duplicate of the calling (parent) process. The exec functions replace the current process image with a new process image.

3. Delta Description: The fork and exec functions accomplish the OSSWG Create Process (2.9.1) and Start Process (2.9.3) functions. That is, processes are started upon creation. The fork and exec functions do not provide the ability to specify process attributes.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 1
   Criteria Name: Create Process
   Requirement Document Paragraph: 2.9.1

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4 (August 10,1990)
   Revision:
   Page: 21-23 (thread creation attributes) and 23-24 (thread creation)
   Paragraph: 3.3.1 (thread creation attributes) and 3.3.2 (thread creation)

   Capabilities Summary: Threads are created with attributes specified by a threads attributes object. The functions pthread_attr_create, pthread_attr_delete, pthread_attr_setstacksize, and pthread_attr_getstacksize are defined for creating, deleting, setting, and examining thread creation attributes. Threads are created via the function pthread_create.

3. Delta Description: P1003.4a satisfies this requirement. The attribute specification method is sufficiently general and extensible.

4. Comments: Two additional functions were added at the October 1990 POSIX meeting: pthread_attr_setcontentionscope and pthread_attr_getcontentionscope. The scope can be local or global.

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 2
   Criteria Name: Terminate Process
   Requirement Document Paragraph: 2.9.2

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 56 (_exit) and 62 (kill)
   Paragraph: 3.2.2 (_exit) and 3.3.2 (kill)

   Capabilities Summary: The _exit function terminates the calling process. The kill function sends a specified signal to a specified process or group of processes. The termination signal is SIGKILL, which cannot be caught or ignored, and whose default action is abnormal termination of the process.

3. Delta Description: In regard to termination of POSIX.1 processes, 1003.1 satisfies this requirement.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 2
   Criteria Name: Terminate Process
   Requirement Document Paragraph: 2.9.2

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 26 (pthread_exit) and 69-80 (thread cancellation)
   Paragraph: 3.3.5 (pthread_exit) and Section 6 (thread cancellation)

   Capabilities Summary: The pthread_exit function terminates the calling thread. In Draft D4, the thread cancellation mechanism was defined to allow a thread to terminate the execution of any other thread in the process in a controlled manner. The function is pthread_cancel. The target thread (i.e., the one being canceled) is allowed to hold cancellation requests pending by setting its cancelability state (via the functions pthread_setcancel and pthread_setasynchcancel) and by creating cancellation points (via the function pthread_testcancel). The target thread can perform application-specific cleanup processing when the notice of cancellation is acted upon. The thread does so by pushing and popping cleanup routines onto its cleanup stack (via the functions pthread_cleanup_push and pthread_cleanup_pop).

3. Delta Description: P1003.4a almost satisfies this requirement via the pthread_exit and the pthread_cancel interfaces, combined with the interfaces for establishing cleanup handlers. The burden of releasing resources is on the writer of the thread cleanup handlers. The notion of disabling cancelability and/or allowing cancelability only at specific cancellation points is the only major incompatibility with this requirement. For purposes of fault tolerance, the NGCR OS must have some sort of absoulte interface to force a thread to terminate (or at least not compete for the CPU) if it refuses to terminate cooperatively. Depending on the scheduling policy, this could perhaps be accomplished via pthread_setprio (to a priority that never executes), but a clean "pthread_abort" interface would be an appropriate portable solution. The consequences of an uncoordinated asynchronous abort of a thread are understood, and such an abort is not required under most circumstances (even in an Ada abort statement); but to deny the requirement is akin to denying that "kill -9" is ever needed in a standard UNIX system.

4. Comments: Note that Draft D4 states that the cleanup routine is executed when the thread terminates normally (via pthread_exit). (As I recall,) thread cancellation was covered in the signal interface during the October 1990 POSIX meeting. So, this requirement needs to be revisited when the next draft becomes available.

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 3
   Criteria Name: Start Process
   Requirement Document Paragraph: 2.9.3

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 49 (fork) and 50 (exec)
   Paragraph: 3.1.1 (fork) and 3.1.2 (exec)

   Capabilities Summary: (Creating) and starting a new process is accomplished via a sequence of two function calls, (1) fork and (2) one of the functions in the exec family. The fork function creates a new (child) process, which is a duplicate of the calling (parent) process. The exec functions replace the current process image with a new process image.

3. Delta Description: The fork and exec functions accomplish the OSSWG Create Process (2.9.1) and Start Process (2.9.3) functions. That is, processes are started upon creation. 1003.1 does not provide separate and explicit Start Process and Stop Process functions.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 3
   Criteria Name: Start Process
   Requirement Document Paragraph: 2.9.3

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 23 (pthread_create), 32 (rationale), 57 (pthread_cond_wait), and 56 (pthread_cond_signal and pthread_cond_broadcast)
   Paragraph: 3.3.2 (pthread_create), 3.5.2.3 (rationale), 5.3.7 (pthread_cond_wait), and 5.3.6 (pthread_cond_signal and pthread_cond_broadcast)

   Capabilities Summary: The P1003.4a function pthread_create does both creation and starting.

3. Delta Description: As stated in the rationale, P1003.4a satisfies this requirement through its thread synchronization mechanisms. The start_routine of the created thread can synchronize by waiting on a condition variable (via pthread_cond_wait), which the start operation will signal. The only disadvantage to this scheme is the additional pair of context switches between thread creation and thread start. While this may be considered as having a negative impact on performance, real-time embedded systems seldom engage in the (already high-cost) act of process creation during time-critical scenarios. Thus, the pthreads approach is adequate.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 4
   Criteria Name: Stop Process
   Requirement Document Paragraph: 2.9.4

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page:
   Paragraph:

   Capabilities Summary:

3. Delta Description: 1003.1 does not provide separate and explicit Start Process and Stop Process functions.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 4
   Criteria Name: Stop Process
   Requirement Document Paragraph: 2.9.4

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page:
   Paragraph:

   Capabilities Summary:

3. Delta Description: 1003.4a does not provide separate and explicit Start and Stop functions. However, such preparation for restart could be accomplished through a combination of POSIX.4 asynchronous event notification (as extended to per-thread), per-thread setjmp/longjmp processing, and pthread_cond_wait. The thread would, of course, need to have anticipated the requirement to stop for potential restart and would have to cooperate. Since restarting of a thread must be a planned activity in the design of a system, the pthreads approach is adequate.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 5
   Criteria Name: Suspend Process
   Requirement Document Paragraph: 2.9.5

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 68
   Paragraph: 3.4.2 (pause)

   Capabilities Summary: The pause function suspends the calling process until delivery of a signal.

3. Delta Description: A process can suspend itself using pause. 1003.1 does not provide explicit interfaces for a process to suspend and resume another process.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 5
   Criteria Name: Suspend Process
   Requirement Document Paragraph: 2.9.5

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 33 (rationale) and 57 (pthread_cond_wait)
   Paragraph: 3.5.2.4 (rationale) and 5.3.7 (pthread_cond_wait)

   Capabilities Summary: The function pthread_cond_wait causes the calling thread to wait on the specified condition variable.

3. Delta description: As noted in the rationale, Draft D4 does not provide explicit interfaces for suspending or resuming a given thread. However, P1003.4a partially satisfies this by providing condition variables on which a thread may suspend itself, and a mechanism for resuming that thread by signaling the condition. The rationale mentions that asynchronous suspension of threads is beyond the scope of the current proposal because it is considered error-prone and would introduce unnecessary complexity because of additional state(s) that would be associated with a thread. The current workaround to the OSSWG requirement would require each suspendable thread to include code allowing it to "voluntarily" suspend itself upon being told to do so via one of several synchronous or asynchronous IPC facilities. This is probably adequate if we allow uncooperative thread to be terminated asynchronously (see Criteria 2 under Service Class 9).

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 6
   Criteria Name: Resume Process
   Requirement Document Paragraph: 2.9.6

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 68
   Paragraph: 3.4.2

   Capabilities Summary: A process that has executed the pause function is suspended until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. For resuming execution, the action must be to execute a signal-catching function. Execution resumes after the signal-catching function returns.

3. Delta Description: 1003.1 does not provide explicit interfaces for a process to suspend and resume another process.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 6
   Criteria Name: Resume Process
   Requirement Document Paragraph: 2.9.6

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 33 (rationale) and 56 (pthread_cond_signal and pthread_cond_broadcast)
   Paragraph: 3.5.2.4 (rationale) and 5.3.6 (pthread_cond_signal and pthread_cond_broadcast)

   Capabilities Summary: The functions pthread_cond_signal and pthread_cond_broadcast are used to wake up one or more threads suspended because they're waiting on a condition variable.

3. Delta Description: As noted in the rationale, Draft D4 does not provide explicit interfaces for suspending and resuming a given thread. However, this requirement can be met indirectly through the condition variable synchronization mechanism. A thread suspended via pthread_cond_wait can be resumed via pthread_cond_signal. Furthermore, Draft D4 goes beyond the stated NGCR OSSWG requirement by providing pthread_cond_broadcast, which could be used to wake several related suspended threads with a single invocation.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 7
   Criteria Name: Delay Process
   Requirement Document Paragraph: 2.9.7

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 69
   Paragraph: 3.4.3

   Capabilities Summary: The sleep function can be used to suspend the calling process for a specified number of seconds.

3. Delta Description: Resolution of seconds may be unacceptable. 1003.1 does not provide an explicit interface for a process to delay another process.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 7
   Criteria Name: Delay Process
   Requirement Document Paragraph: 2.9.7

2. POSIX Reference
   Document Number: P1003.4
   Draft: D9
   Revision:
   Page: 126 (nanosleep) and 124 (abstimer)
   Paragraph: 8.4.5 (nanosleep) and 8.4.4 (abstimer)

   Capabilities Summary: A process can delay itself for a specified amount of time, given in nanoseconds, via the nanosleep function. The actual resolution supported by the nanosleep function may not be nanoseconds; the resolution can be obtained via the ressleep function. A process can delay itself until an absolute time via the abstimer function (specified as a one-shot timer).

3. Delta Description: 1003.4 does not provide an explicit interface for a process to delay another process. However, a process could use an IPC mechanism to tell another process to voluntarily delay itself.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 7
   Criteria Name: Delay Process
   Requirement Document Paragraph: 2.9.7

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 21 (nanosleep)
   Paragraph: 3.3 (nanosleep)

   Capabilities Summary: The POSIX.4 nanosleep function is redefined so that only the calling thread (and not process) is suspended.

3. Delta Description: P1003.4a does not provide an explicit interface for a thread to delay until an absolute time. The function abstimer is not explicitly redefined to work on a per-thread basis; however, in the definition of thread on p. 10 of Draft D4, it is stated that each thread has its own state of any timer. (What about per-process interval timers? Do they become per-thread?) P1003.4a does not provide an explicit interface for a thread to delay another thread. Thus, one thread forcing another thread to delay would require cooperative code in each thread. The thread to be delayed would have to recognize and act upon an IPC request to delay itself.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 8
   Criteria Name: Interprocess Communication
   Requirement Document Paragraph: 2.9.8

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 62 (kill, or Send a Signal to a Process), 109 (pipe), and 83-120 (files, directories, and I/O)
   Paragraph: 3.3.2 (kill), 6.1 (pipe), and 5 and 6 (files, directories, and I/O)

   Capabilities Summary: The kill function sends a specified signal to a specified process. A pipe is an interprocess communication channel. The pipe function creates a pipe. Then, one process can write to the pipe, and another process can read from the pipe. Data is buffered in the pipe, and it is read on a FIFO basis.

3. Delta Description: These mechanisms provide limited IPC capabilities. 1003.4 adds binary semaphores, shared memory, interprocess message passing, realtime files, asynchronous I/O, and synchronized I/O.

4. Comments: 1003.1 provides interfaces for file and directory management and for I/O. Descriptions of these interfaces are not offered here, but instead are presumed to be covered under Service Class 6 (File Interfaces).

1. Service Class Number: 9
Service Class Name: Process Management Interfaces
Criteria Number: 8
Criteria Name: Interprocess Communication
Requirement Document Paragraph: 2.9.8

2. POSIX Reference
Document Number: P1003.4
Draft: D9
Revision:
Page: 21-32 (binary semaphores); 43-60 (shared memory); 137-176 (message passing); 177-236 (realtime files, asynchronous I/O, synchronized I/O)
Paragraph: Section 3 (binary semaphores); Section 5 (shared memory); Section 9 (message passing); Sections 12, 11, and 10 (realtime files, asynchronous I/O, synchronized I/O)

Capabilities Summary:

- Binary semaphores. These are adopted as the P1003.4 high-performance process synchronization mechanism. Binary semaphores are referred to as binary semaphore special files; they are objects named within the file system name space. Semaphores can be persistent (state preserved from last close to first open) or non-persistent (always in unlocked state at first open). Operations are semwait (P operation), semifwait (conditional P operation), sempost (V operation), and semifpost (conditional V operation). The mksem operation creates a binary semaphore special file, and the unlink removes it. The open and close operations apply to the use of binary semaphore special files.

- Shared memory. P1003.4 provides shared memory as a mechanism for high-performance process synchronization and communication. Shared memory is supported through the mechanism of shared memory special files, objects named within the file system name space. Shared memory special files can be persistent (contents preserved across the "unreferenced" state) or non-persistent (contents undefined when file becomes unreferenced). Functions include mkshm (create shared memory special file), unlink (remove file), open, close, shmmap (map shared memory into process's address space), and shmunmap (unmap). A shared memory special file can be opened read only, write only, or read/write.

- Message passing. P1003.4 provides message passing as a mechanism for interprocess communication. Message passing is supported through message queue special files, objects names within the file system name space. The functions mkmq (make message queue special file), unlink (remove file), open, and close are provided. Message queue attributes for flow and resource control can be set and examined via the functions mqsetattr and mqgetattr. Messages can be sent synchronously or asynchronously via the function mqsend and the use of flags to indicate type of sending; messages can be received synchronously or asynchronously via the function mqreceive and the use of flags. Short messages (a pointer's worth of data) can be sent and received as event data via the functions mqputevt and mqgetevt. Message data buffers can be allocated and freed via the functions msgalloc and msgfree. Messages (either all messages sent by a specified process or all messages) can be purged from the message queue via the function mqpurge. The process identifier of the sending process of a specified message can be obtained via the function mqgetpid.

3. Delta Description: When combined, 1003.1 and P1003.4 meet the stated requirement, which is to offer a wide variety of services for processes to exchange information.

4. Comments: 1003.4 provides interfaces for "realtime files" and and for asynchronous I/O and "synchronized I/O". Descriptions of these interfaces are not offered here, but instead are presumed to be covered under Service Class 6 (File Interfaces).

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 8
   Criteria Name: Interprocess Communication
   Requirement Document Paragraph: 2.9.8

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 20 (binary semaphores), 21 (message passing), 91-109 (signals)
   Paragraph: 3.3 (binary semaphores), 3.3 (message passing) and I/O), Section 9 (signals)

   Capabilities Summary:

   - Shared memory. Threads within a process share memory by default. Mutexes and condition variables are provided as synchronization mechanisms.

   - Binary semaphores. The function semwait is redefined so that only the calling thread is suspended.

   - Message passing. The functions mqsend and mqreceive are redefined so that only the calling thread is suspended.

   - Signals. The signal mechanism is redefined to optionally work on a per-thread basis.

3. Delta Description: P1003.4 plus P1003.4a provide a more than adequate foundation for IPC among heavyweight and/or lightweight processes.

4. Comments: The discussion of files and I/O is presumed to be covered in the context of Service Class 6 (File Interfaces). The signal facilities underwent a major revision at the October 1990 POSIX meeting, so this requirement needs to be revisited when the next draft of P1003.4a becomes available.

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 9
   Criteria Name: Examine Process Attributes
   Requirement Document Paragraph: 2.9.9

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 71
   Paragraph: Section 4 (Process Environment)

   Capabilities Summary: 4.1.1: Get process and parent process ids of calling process. 4.2.1: Get real and effective user and group ids of calling process. 4.2.3: Get supplementary group ids of calling process. 4.2.4: Get name of user associated with calling process. 4.3.1: Get process group id of calling process. 4.7.1: Get id of current controlling terminal for calling process.

3. Delta Description: 1003.1 does not provide an explicit interface for a process to examine the attributes of another process.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 9
   Criteria Name: Examine Process Attributes
   Requirement Document Paragraph: 2.9.9

2. POSIX Reference
   Document Number: P1003.4
   Draft: D9 (December 1989)
   Revision:
   Page: 67 (get sched priority) and 69 (get sched policy)
   Paragraph: 6.5.2 (priority) and 6.5.4 (policy)

   Capabilities Summary: The getprio and getscheduler functions return the priority and scheduling policy of a specified process.

3. Delta Description:

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 9
   Criteria Name: Examine Process Attributes
   Requirement Document Paragraph: 2.9.9

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 21 (thread creation attributes), 26 (pthread_self), and 41 (thread creation scheduling attributes)
   Paragraph: 3.3.1 (thread creation attributes), 3.3.6 (pthread_self), and 4.3.1 (thread creation scheduling atttibutes)

   Capabilities Summary: P1003.4a provides a thread attributes object that is used to create threads. As noted in the rationale, the attributes object is purposely defined as an opaque type to facilitate extensibility. In Draft D4, functions to examine the stacksize attribute in a specified attributes object are defined (pthread_attr_getstacksize). Functions to examine scheduling attributes are also defined (i.e., pthread_attr_getinheritsched, pthread_attr_getsched, and pthread_attr_getprio). In addition, functions to examine the dynamic values of scheduling attributes for a specified thread are defined (i.e., pthread_getscheduler, pthread_getprio). The function pthread_self can be used to examine the thread ID of the calling thread.

3. Delta Description: P1003.4a seems to satisfy this requirement.

4. Comments: Note that 4 new functions are required for each new attribute to be supported by an implementation, two to get and set static attributes in the attributes object and two to get and set dynamic attributes. The next draft of P1003.4a should be reviewed when it becomes available, since some revisions concerning attribute objects were made during the October 1990 POSIX meeting.

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 10
   Criteria Name: Modify Process Attributes
   Requirement Document Paragraph: 2.9.10

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 71
   Paragraph: Section 4 (Process Environment)

   Capabilities Summary: 4.2.2: Set user and group ids of calling process. 4.3.2: Create session and set process group id for the calling process.

3. Delta Description: 1003.1 does not provide an explicit interface for a process to modify the attributes of another process.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 10
   Criteria Name: Modify Process Attributes
   Requirement Document Paragraph: 2.9.10

2. POSIX Reference
   Document Number: P1003.4
   Draft: D9
   Revision:
   Page: 65 (set sched priority) and 68 (set sched policy)
   Paragraph: 6.5.1 (priority) and 6.5.3 (policy)

   Capabilities Summary: The setprio function sets the priority of a specified process to a specified value. The setscheduler function sets the scheduling policy and priority of a specified process to specified values.

3. Delta Description: P1003.4 provides only limited capabilities here.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 10
   Criteria Name: Modify Process Attributes
   Requirement Document Paragraph: 2.9.10

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 21 (thread creation attributes) and 41 (thread creation scheduling attributes)
   Paragraph: 3.3.1 (thread creation attributes) and 4.3.1 (thread creation scheduling atttibutes)

   Capabilities Summary: P1003.4a provides a thread attributes object that is used to create threads. As noted in the rationale, the attributes object is purposely defined as an opaque type to facilitate extensibility. In Draft D4, a function to modify the stacksize attribute in a specified attributes object is defined (pthread_attr_setstacksize). Functions to modify scheduling attributes are also defined (i.e., pthread_attr_setinheritsched, pthread_attr_setsched, and pthread_attr_setprio). In addition, functions to dynamically modify the values of scheduling attributes for a specified thread are defined (i.e., pthread_setscheduler, pthread_setprio).

3. Delta Description: P1003.4a seems to satisfy this requirement.

4. Comments: Note that 4 new functions are required for each new attribute to be supported by an implementation, two to get and set static attributes in the attributes object and two to get and set dynamic attributes. The next draft of P1003.4a should be reviewed when it becomes available, since some revisions concerning attribute objects were made during the October 1990 POSIX meeting. Question: why is there no pthread_setstacksize?

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 11
   Criteria Name: Examine Process Status
   Requirement Document Paragraph: 2.9.11

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: None
   Paragraph:

   Capabilities Summary:

3. Delta Description: 1003.1 does not provide explicit interfaces for examining process status in regard to state of the process (terminated, stopped, suspended, blocked, etc.).

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 11
   Criteria Name: Examine Process Status
   Requirement Document Paragraph: 2.9.11

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: None
   Paragraph:

   Capabilities Summary:

3. Delta Description: 1003.4a does not provide explicit interfaces for examining thread status in regard to state of the thread (terminated, stopped, suspended, blocked, etc.).

4. Comments: A non-blocking alternative to pthread_join might parially or fully satisfy this requirement. An explicit interface for this purpose would be better.

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 12
   Criteria Name: Process Identification
   Requirement Document Paragraph: 2.9.12

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 33
   Paragraph: Definition of process ID

   Capabilities Summary: Each process in the system is uniquely identified during its lifetime by a positive integer. "System" is defined to be implementation of 1003.1. Extent of distribution possible in a system is not defined.

3. Delta Description: 1003.1 does not explicitly address distributed systems.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 12
   Criteria Name: Process Identification
   Requirement Document Paragraph: 2.9.12

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 10
   Paragraph: Definition of thread ID

   Capabilities Summary: According to Draft D4, each thread in a process is uniquely identified during its lifetime by its thread ID. Although implementations may have thread IDs which are unique in a system, applications should only assume that thread IDs are usable and unique within a single process.

3. Delta Description: Threads are guaranteed to be uniquely identified only within a process. Processes are uniquely defined within a system. So, a thread is uniquely defined by its process ID and thread ID pair. Distributed systems are not explicitly considered.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 13
   Criteria Name: Save/Restart Process
   Requirement Document Paragraph: 2.9.13

2. POSIX Reference: None
   Document Number:
   Draft:
   Revision:
   Page:
   Paragraph:

   Capabilities Summary:

3. Delta Description:

4. Comments: At this time, these interfaces are considered to be out of POSIX scope.

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 13
   Criteria Name: Save/Restart Process
   Requirement Document Paragraph: 2.9.13

2. POSIX Reference: None
   Document Number:
   Draft:
   Revision:
   Page:
   Paragraph:

   Capabilities Summary:

3. Delta Description: At this time, these interfaces are considered to be out of POSIX scope.

4. Comments: There are no interfaces by which one thread may query the state of another thread, nore is there any apparent method for a thread to be asynchronously restarted. A cooperative thread could be set up to checkpoint itself and restart from a checkpoint of its own design upon receipt of a synchronous or asynchronous IPC request to do so. However, it is the nature of embedded mission criticial systems that need for restarting a process from a saved state usually occurs under circumstances when the to-be-restarted process is less than cooperative, i.e., in fault tolerant recovery situations. Therefore, it is imperative that the pthreads standard address the issue ofportable interfaces to thread save/restore capabilities. This may require spcification of an opaque type to hold an abstract thread state.

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 14
   Criteria Name: Program Management Function
   Requirement Document Paragraph: 2.9.14

2. POSIX Reference
   Document Number: IEEE Std 1003.1-1988
   Draft:
   Revision:
   Page: 32
   Paragraph: Definition of process

   Capabilities Summary: A process is an address space and single thread of control that executes within that address space, and its required system resources. Each process is a member of a process group, and each process group is a member of a session.

3. Delta Description: 1003.4a provides for multiple threads of control within a process. Then, the process corresponds to an Ada program, and the threads to Ada tasks.

4. Comments:

1. Service Class Number: 9
   Service Class Name: Process Management Interfaces
   Criteria Number: 14
   Criteria Name: Program Management Function
   Requirement Document Paragraph: 2.9.14

2. POSIX Reference
   Document Number: P1003.4a
   Draft: D4
   Revision:
   Page: 29-33
   Paragraph: 3.5 (Rationale Relating to Thread Management)

   Capabilities Summary: P1003.4a provides pthreads as a mechanism for enabling multiple flows of control to exist within a single POSIX.1 process.

3. Delta Description: 1003.4a satisfies this requirement by providing a concurrency abstraction (pthreads) which runs within the context of a POSIX.1 process. The process, with its loosely coupled IPC mechanisms, limited shared memory capability, and inherently heavyweight context matches the requirement for a "program" well (and that is its most frequent use in typical timesharing UNIX systems), while pthreads provides the concurrency model within the context of a process. In regard to Ada, an Ada program corresponds to a POSIX.1 process, and an Ada task to a pthread.

4. Comments:

References: 1003.1, 1003.4 (D9), 1003.4a (D4), the NGCR OSSWG POSIX cross matrix, the NGCR OSSWG evaluation comments and rationales, and Frank Prindle's white paper "Pthreads Meets NGCR"

Revised Data Sheets for Service Class 9, Criteria 11 and 12 (September 1991)

To be included in Appendix A of the DELTA Document for the Next Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline

Submitted by Karen Gordon (IDA)

9.11

1. Service Class Number :9
Service Class Name :Process Management Interfaces
Criteria Number :11
Criteria Name :Examine Process Status
Requirement Document Paragraph :20.9.11

2. POSIX Reference
Document Number :IEEE Std 1003.1-1990
Draft :
Revision :
Page : 47, 49, 231
Paragraph : 3.2.1

Capabilities Summary: The wait() and waitpid() functions allow the calling process to obtain status information on child processes-- whether they have terminated or stopped.

3. Delta Description: The wait() and waitpid() functions provide limited status (terminated, stopped, and why (e.g., caused by which signal)) on limited processes (child processes). Richer status information is required. The ability to examine status of general processes (i.e., non-children) is required.

4. Comments:

(1) p. 47: A non-blocking version of waitpid() is available; it is indicated by a "WNOHANG" flag in the options argument.

(2) p. 49: "An implementation may define additional circumstances under which wait() or waitpid() reports status."

(3) p. 231: "Nothing in POSIX.1 prevents an implementation from providing extensions that permit a process to get status from a grandchild or any other process."

(4) The UNIX mechanism for obtaining process status is the "ps" command. Likely places for the "ps" command to appear in POSIX include 1003.2 and 1003.7 (note: check with these groups and their drafts).

9.11

1. Service Class Number :9
Service Class Name :Process Management Interfaces
Criteria Number :11
Criteria Name :Examine Process Status
Requirement Document Paragraph :20.9.11

2. POSIX Reference
Document Number :P1003.4a
Draft :D5
Revision :
Page : 29, 30
Paragraph : 3.4.3

Capabilities Summary: p. 29: "The pthread_join function suspends execution of the calling thread until the target thread terminates." This interface provides limited status information--whether a thread has terminated.

3. Delta Description: Richer status information is required.

4. Comments: A non-blocking alternative to pthread_join might partially or fully satisfy this requirement. (See non-blocking alternative to waitpid for process.) As suggested on p. 30 of P1003.4a/D5, condition variables could be used to achieve effect of non-blocking join. An explicit interface for this purpose would be better.

9.12

1. Service Class Number :9
Service Class Name :Process Management Interfaces
Criteria Number :12
Criteria Name :Process Identification
Requirement Document Paragraph :20.9.12

2. POSIX Reference
Document Number :IEEE Std 1003.1-1990
Draft :
Revision :
Page : 18
Paragraph :Definition of process ID

Capabilities Summary: Each process in the system is uniquely identified during its lifetime by a positive integer. "System" is defined to be an implementation of 1003.1.

3. Delta Description: None

4. Comments:

(1) A 1003.1 system could span multiple processors (connected by shared memory or network); such a multiprocessor or distributed operating system implementation of a 1003.1 "system" is not precluded. In this case, the process id would be unique across all the platforms composing the system.

(2) Many aspects of distribution (naming, communication, management, etc., across multiple POSIX "systems") are covered under networking services (e.g., 1003.17 (Directory) for naming, 1003.12 (PII) for communication) and system administration (i.e., 1003.7 for management).

(3) From p. 18, "In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID shall not be reused by the system until the process group lifetime ends."

(4) From p. 18, regarding process lifetime: Ends only after a process terminates AND the parent process executes a wait() or waitpid() for the process. That is, a process is in an inactive state after it has terminated but before it has been "wait"ed for by its parent; during this inactive period, its process id cannot be reused. So, process ID uniqueness can be preserved by refraining from the execution of the "wait" or "waitpid".

9.12

1. Service Class Number :9
Service Class Name :Process Management Interfaces
Criteria Number :12
Criteria Name :Process Identification
Requirement Document Paragraph :20.9.12

2. POSIX Reference
Document Number :P1003.4a
Draft :D5
Revision :
Page :10
Paragraph :Definition of thread ID

Capabilities Summary: According to Draft D5, each thread in a process is uniquely identified during its lifetime by its thread ID. (Although implementations may have thread IDs which are unique in a system, applications should only assume that thread IDs are usable and unique within a single process.) In the context of a system running multiple processes, any thread in the system can be uniquely identified by a pair of IDs--the first component of the pair being the process ID of the process in which the thread resides, and the second component being the thread ID of the thread itself.

3. Delta Description: None

4. Comments: From p. 11: "A conforming implementation is free to reuse a thread ID after the thread terminates and pthread_detach() is called for that thread." So, by not detaching a thread, thread ID uniqueness throughout the lifetime of the process can be guaranteed. Process ID uniqueness can be guaranteed as indicated above.

Resolution of Deltas for Service Class 9, Criteria 11, 12, and 13 (September 1991)

To be included in Section 6 of the DELTA Document for the Next Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline

Submitted by Karen Gordon (IDA)

9.0 Process Management Interfaces

9.11 Examine Process Status

Requirement: The OSIF shall provide the ability for processes to examine the current status of a particular process. This requirement would be classified as (a) Required. Note that status here is not intended to include cumulative execution time; the capability to obtain cumulative execution time is covered as Criteria 3 in Service Class 13 (Synchronization and Scheduling).

Description of Deltas: The wait() and waitpid() functions provide limited status (terminated, stopped, and why (e.g., caused by which signal)) on limited processes (child processes). Richer status information is required. The ability to examine status of general processes (i.e., non-children) is required.

Resolution Alternatives:

1) Enhance existing 1003.1 wait() and waitpid() interfaces to include this capability - Extensions of wait() and waitpid() to provide richer status information and to allow status querying to general processes are discussed in 1003.1, but not included in the standard. It is unlikely that consensus to include the extensions could be achieved.

2) Incorporate UNIX "ps" command functionality into a POSIX standard. The functionality should be incorporated as a system call, and also as a command ("ps" is available only as a command in present UNIX implementations).

Recommendation: The 1003.7 and 1003.2 drafts should be reviewed to determine whether "ps" is on the agenda. The 1003.2 and/or 1003.7 groups should be approached with a proposal to include the capability for examining process status in their drafts (if not already on the agenda of one of the groups). The system call version of "ps" might need to be placed in another draft (possibly one of Working Group 1003.4's drafts).

9.11 Examine Thread Status

Requirement: The OSIF shall provide the ability for threads to examine the current status of a particular thread. This requirement would be classified as (a) Required. Note that status here is not intended to include cumulative execution time; the capability to obtain cumulative execution time is covered as Criteria 3 in Service Class 13 (Synchronization and Scheduling).

Description of Deltas: The pthread_join function provides limited status information--whether a thread has terminated. Richer status information is required.

A non-blocking alternative to pthread_join might partially or fully satisfy this requirement. (See non-blocking alternative to waitpid for process.) As suggested on p. 30 of P1003.4a/D5, condition variables could be used to achieve effect of non-blocking join. An explicit interface for this purpose would be better.

Resolution Alternatives:

1) Frank Prindle's proposal for pthread_timedjoin (as part of timeouts for blocking services proposal for 1003.4b).

2) Investigate extending UNIX "ps" command functionality to threads and incorporating into a POSIX standard (probably one in the P1003.4 series, since threads are addressed in the 1003.4 Working Group).

Recommendation: Alternative 1) (Frank's proposal) meets part of the requirement. To get the richer status information, alternative 2) should also be pursued.

9.12 Process (Thread) Identification

Requirement: The OSIF shall support the unambiguous identification of processes (threads). This requirement would be classified as (a) Required.

Description of Deltas: None (See revised data sheets).

9.13 Save/Restart Process

Requirement: The OSIF shall support the ability for processes to be restarted from a saved state. This requirement would be classified as (a) Required.

Description of Deltas: At this time, these interfaces are considered to be out of POSIX scope.

Note: check the P1003.1a draft; it was reported that checkpointing may currently be an agenda item there.

Resolution Alternatives:

1) Investigate checkpointing/restarting of processes and threads, possibly in the context of a broader OSSWG fault tolerance proposal. Consider 1003.7 and 1003.4 as forums for making proposals.

2) Levy the requirements and the OSIF general requirements on vendors but do not provide a standard as such - This alternative relies on vendors to develop some commercial existing practice in this area on which to potentially standardize at a later date.

Recommendation: Alternative 1) is recommended, while it is recognized that program managers can always resort to Alternative 2). Checkpointing a process that is communicating with others or checkpointing a thread (that is sharing memory with other threads) seems to be difficult and demands further study.

Resolution of Deltas for Service Class 13, Criteria 7 and 11 (September 1991)

To be included in Section 6 of the DELTA Document for the Next Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline

Submitted by Karen Gordon (IDA)

13.0 Synchronization and Scheduling Interfaces

13.7 Periodic Scheduling

Requirement: The OSIF shall provide for the periodic scheduling of a process (thread). This requirement would be classified as (a) Required.

Description of Deltas: From Delta Document data sheets: "Need guarantee that there will be no unreasonable delay in the runnability of a process (thread) once a specified periodic time has arrived."

Resolution Alternatives:

1) Define what is meant by "no unreasonable delay" and enhance existing POSIX interfaces to include this capability.

2) Accept the POSIX approach--performance metrics--and require vendors to report values for the performance metrics cited in Section 8.4 of 1003.4/D10 (pp. 147-149) (in particular, clock/timer granularity and timer expiration service latencies) and also in Section 7.4 (event dispatch latency).

Recommendation: Alternative 2). Since the definition of "no unreasonable delay" is implementation-dependent and application-dependent, the POSIX approach of specifying performance metrics and requiring their values to be reported is recommended. (Note: the delta for 13.7 (periodic scheduling) could be viewed as "none".)

13.11 Precise Scheduling (Jitter Management)

Requirement: The OSIF shall provide the ability for an application to indicate to the scheduler an exact (actually, within some tolerance) specified time for starting a process (thread). This requirement would be classified as (a) Required.

Description of Deltas: From Delta Document data sheets: "Need guarantee that there will be no unreasonable delay in the runnability of a process (thread) once a specified precise time has arrived or precise interval has expired."

Resolution Alternatives:

1) Define what is meant by "no unreasonable delay" and enhance existing POSIX interfaces to include this capability.

2) Accept the POSIX approach--performance metrics--and require vendors to report values for the performance metrics cited in Section 8.4 of 1003.4/D10 (pp.147-149) (in particular, clock/timer granularity, clock jitter, and timer expiration service latencies) and also in Section 7.4 (event dispatch latency).

Recommendation: Alternative 2). Since the definition of "no unreasonable delay" is implementation-dependent and application-dependent, the POSIX approach of specifying performance metrics and requiring their values to be reported is recommended. (Note: the delta for 13.11 (precise scheduling) could be viewed as "none".)