OZIX Technical Summary

Version: 0.5

30 October 1989

Issued by: DECwest Engineering

digital™

This document was prepared using VAX DOCUMENT, Version 1.2

Acknowledgements

This document represents the original work of many individuals on the OZIX project. Key contributors include the following.

Document Planning and Production

Marilyn Fries, John Penney, Gordon Furbush, Bill Talcott

Chapter Authors

Introduction	Gordon Furbush Jeff Havens
User Environment and Application Environment	Myles Connors
System Administration	Mark Ditto
Resource/Fault Management	David Walp
Security	Jim Schirmer
Internationalization	Claire Cockcroft
Base System Architecture	Chris Saether
I/O System	Sumanta Chatterjee Debbie Girdler
Terminal Subsystems	Sumanta Chatterjee
Network Implementation Architecture	OZIX Networks Group
System Installation and System Directory Layout	Mark Ditto
Performance Engineering	Pete Benoit Mike Peterson
Software Quality and Testing Strategy	Jan D'Addamio

TABLE OF CONTENTS

Preface v

CHAPTER 1 INTRODUCTION 1

1.1 The OZIX Project 1

1.2 Building the Foundation of a Modern Operating System 2

1.3 Building an Open System 2

1.4 The OZIX Architecture: A Structured Approach 2

1.5 User and Application Support 5

1.6 Transaction Processing Primitives 6

1.7 Process Support 6

1.8 Memory Management Interface 7

1.9 I/O Support 7

1.10 Distributed Computing Environment 7

1.11 System Administration 8

1.12 Built-in Performance Analysis 8

1.13 Outline 9

CHAPTER 2 USER ENVIRONMENT AND APPLICATION ENVIRONMENT 11

2.1 Types of OZIX Users 11
2.1.1 End Users 12
2.1.2 Application Programmers 13
2.1.3 System Administrators 13

2.2 Ways Users Access OZIX 14
2.2.1 Logging in to OZIX 15
2.2.2 Executing Applications From a Remote System 16
2.2.3 Executing Applications Using the Batch System 16
2.2.4 Accessing Data on OZIX From a Remote System 17

2.3 The OZIX End User Environment 17
2.3.1 Standard Commands and Utilities 18
2.3.2 OZIX Shells 18
2.3.3 Character-Cell-Terminal User Interface 19
2.3.4 Workstation User Interface 19
2.3.4.1 Invoking Window-based Applications 20

2.3.5 The End User View of OZIX Security 21
2.3.5.1 The End User View of Access Control Lists 22
2.3.5.2 The End User View of the TCB Shell 22
2.3.6 The End User View of OZIX Internationalization Facilities 23
2.3.6.1 Internationalization and Standard Commands and Utilities 23
2.3.6.2 Internationalization and Terminal Driver Support 24
2.3.6.3 How End Users Control Internationalization Support 25

2.4 The OZIX Application Development Environment 25
2.4.1 OZIX Programming Tools 25
2.4.1.1 Standard Commands and Utilities 25
2.4.1.2 Digital Added Value Commands and Utilities 26
2.4.1.3 Third-Party Tools 26
2.4.2 The Application Execution Environment 26
2.4.2.1 Standard Application Program Interfaces 28
2.4.2.2 Digital Added-Value Application Program Interfaces 29
2.4.2.3 Third-Party Application Program Interfaces 30
2.4.2.4 OZIX Added-Value Application Program Interface Technology 30

2.5 OZIX System Administrator Environment 32
2.5.1 Standard System Administration Tools 32
2.5.2 OZIX Added Value 33

2.6 Summary 34

CHAPTER 3 SYSTEM ADMINISTRATION 35

3.1 Introduction 35

3.2 Problem Summary 35

3.3 Goals 35

3.4 OZIX System Administration Structure 36
3.4.1 Manageable Object 36
3.4.2 Management Backplane 40
3.4.3 User Presentation 41

3.5 Interoperability 42

3.6 OZIX System Administration Component Overview 43

CHAPTER 4 RESOURCE/FAULT MANAGEMENT 47

4.1 Introduction 47

4.2 Goals 47

4.3 Overview 47

4.4 Manageable Objects 50

4.5 Error Detection 50

4.5.1 Hardware Error Detection 50
4.5.1.1 Unreported Hardware Errors 51
4.5.1.2 Hardware Design Faults 51
4.5.2 Software Error Detection 51
4.6 Error Log Writer 52
4.7 Error Report Generator 52
4.8 Error Monitor 53
4.9 Resource Manager 54
4.10 Recovery Mechanisms 54
4.11 User Interfaces for Resource/Fault Management 55
4.12 Crash Dump Writer 55
4.13 Crash Dump Analyzer Description 56
CHAPTER 5 SECURITY 57
5.1 Scope 57
5.2 OZIX Security Goals 57
5.3 Basic Security and Integrity Goals 58
5.3.1 Implementing the Basic Protections 59
5.3.1.1 Identifying the Fundamental Entities 59
5.3.1.1.1 Labeling of Subjects and Objects 60
5.3.1.1.1.1 Unique Security Identity 60
5.3.1.1.1.2 Sensitivity Levels 60
5.3.1.1.1.3 Sensitivity Compartments 60
5.3.1.1.1.4 Integrity Levels 60
5.3.1.1.1.5 Integrity Compartments 61
5.3.1.1.1.6 Security Rings 61
5.3.1.2 Ensuring Isolation 62
5.3.1.3 Granting Access to Objects 63
5.3.1.4 Controlling Usage of Resources 63
5.3.1.5 Trusting Code 63
5.3.1.6 Recording Access Events 63
5.3.2 Obtaining Certification 64
5.3.2.1 Development 64
5.3.2.1.1 Methodology 64
5.3.2.1.2 Configuration Management 64
5.3.2.2 Required Security Features 65
5.3.2.3 Documentation 65
5.3.2.4 Verification 66
5.3.2.5 Testing 66
5.3.2.6 NCSC Approval 66

5.3.2.6.1 NCSC Penetration Testing 66

5.4 Extended Security and Integrity 66
5.4.1 Alternate Models 67
5.4.2 Alternate Authenticators 67
5.4.3 Enforcement Levels 67
5.4.4 Features to Support Secure Applications 68
5.4.4.1 Security Classes 68
5.4.4.2 Access Stacks 68
5.4.4.3 Ring Brackets 68
5.4.4.4 Identity Stack 69
5.4.5 Distributed Security 69

5.5 Global Goals 69

5.6 Examples of Security and Integrity Models 70
5.6.1 Department of Defense Security Model 70
5.6.2 Commercial Access Control Example 71

5.7 Related Reading 74

CHAPTER 6 INTERNATIONALIZATION 75

6.1 Introduction 75
6.1.1 OZIX Description 75
6.1.2 Terminology Definition 75

6.2 Goals 76

6.3 Design Model 76
6.3.1 Problem Statement 76
6.3.1.1 X/OPEN Internationalization 77
6.3.1.2 Multiple Octet Character Set 77
6.3.1.3 Compound String 78
6.3.1.4 Producing International Products (PIP) International Product Model 79
6.3.1.5 Unified Text Representation (UTR) 79

6.4 OZIX International Components 80
6.4.1 Commands 80
6.4.2 Libraries 80
6.4.3 Terminal Services 81
6.4.4 Base System 81
6.4.5 Messaging 81
6.4.6 Programming Tools 81
6.4.7 File System 82

6.5 Additional Internationalization Support 82
6.5.1 Documentation 82
6.5.2 OZIX Release Strategy 82
6.5.3 Training, Support and Service 83

CHAPTER 7 BASE SYSTEM ARCHITECTURE 85
7.1 Introduction 85
7.2 OZIX Base System Architecture Goal 85
7.3 Base System Architecture Structure 85
7.4 OZIX Base System Architecture Functional Modules 86
7.4.1 OZIX Subsystems 87
7.4.1.1 Subsystem Procedures 87
7.4.1.1.1 Subsystem Procedure Calls (SPC) 88
7.4.1.1.2 Gates 89
7.4.1.1.3 Implicit and Explicit Subsystem Procedure Calls 89
7.4.1.2 Subsystem Execution Context 89
7.4.2 OZIX Nub 90
7.5 OZIX Base System Architecture Conceptual Models 90
7.5.1 Executor Model 90
7.5.2 Virtual Memory Model 91
7.5.2.1 Physical to Logical Memory Mapping 92
7.5.2.2 Physical Memory Management 92
7.6 Summary of OZIX Base System Architecture Features 94
CHAPTER 8 I/O SYSTEM 97
8.1 Introduction 97
8.1.1 Goals 98
8.2 Attribute-based Allocation Architecture 99
8.3 File System 100
8.3.1 API Subsystems 101
8.3.2 File Name Subsystem 101
8.3.3 Fault-recoverable Dataspace Subsystem 102
8.3.4 Features Provided by the File System 102
8.3.5 Backup Strategy 102
8.4 Mass Storage Subsystems 103
8.4.1 Mass Storage Components 104
8.4.1.1 Structuring Containers 104
8.4.1.2 Device Drivers 105
8.4.2 Security 107
8.4.3 Management 107
8.4.3.1 System Startup Functions 108
8.4.3.2 Running System 108

CHAPTER 9 TERMINAL SUBSYSTEMS 109

9.1 Introduction 109
9.1.1 Goals 109

9.2 Terminal Components 110

9.3 Security 112

9.4 Terminal Component Management 112

CHAPTER 10 NETWORK IMPLEMENTATION ARCHITECTURE 113

10.1 Introduction 113

10.2 Goals 113

10.3 Strategy 114

10.4 Performance 115

10.5 Architectural Overview 115

10.6 Internet Components 115
10.6.1 Application Layer Code 117
10.6.2 Transmission Control Protocol (TCP) Module 117
10.6.3 User Datagram Protocol (UDP) Module 117
10.6.4 Internet Protocol (IP), Internet Control Message Protocol (ICMP), and Internet Group Management Protocol (IGMP) Modules 117
10.6.5 Routing Database Maintenance 118
10.6.6 Address Resolution Protocol (ARP) and Proxy ARP Modules 118

10.7 DECnet/OSI Components 118
10.7.1 Application Level Components 119
10.7.1.1 File Transfer and Access Management (FTAM) 119
10.7.1.2 Virtual Terminal Protocol (VTP) 119
10.7.1.3 DECnet/OSI Copy Program (DCP) and the File Access Listener (FAL) Daemon 119
10.7.1.4 dlogin and the dlogind daemon 120
10.7.1.5 Distributed Name Service (DNS) Clerk 120
10.7.1.6 Digital Time Synchronization Service (DTSS) Client 120
10.7.2 Session Layer Components 120
10.7.2.1 DNA Session 120
10.7.2.2 OSI Association Control Service Element (ACSE), Presentation, and Session 120
10.7.3 Transport Layer Component 120
10.7.4 DNA Network Layer 120

10.8 Management Components 120
10.8.1 The Backplane 121
10.8.2 The Director 121
10.8.3 The Agent 121
10.8.4 The Event Dispatcher 122

10.9 Network File System 122
10.9.1 Sun Remote Procedure Call 122
10.9.2 NFS Client Subsystem 122
10.9.3 NFS Server Subsystem 122
10.9.4 NFS Locking Server Subsystem 122
10.9.5 NFS Management 123
10.9.6 Techniques used for NFS Performance 123
10.9.7 Highly Available File Service 123

10.10 DEC Remote Procedure Call 123
10.10.1 Network Interface Definition Language (NIDL) Compiler 124
10.10.2 The RPC Runtime Library 124
10.10.3 The Location Broker 125

10.11 Network Interconnects 125

10.12 IBM Interconnect Components 125
10.12.1 Generic IBM Interconnect Components 125
10.12.2 Ported IBM Interconnect Components 126
10.12.2.1 Booting the SNA Gateway 126
10.12.2.2 SNA Gateway Management 126
10.12.2.3 SNA Gateway Monitoring 126
10.12.3 Access to OZIX via SNA 126

10.13 Application Programming Interface (API) 126

10.14 Network Security 127
10.14.1 Philosophy 127
10.14.2 The Trusted Computing Base (TCB) Boundary 127
10.14.3 Performance Options and Tradeoffs 127

10.15 Services 128
10.15.1 Authentication Services 128
10.15.2 Name/Directory Services 128
10.15.3 Mail Services 128
10.15.4 Boot Services 128

10.16 Network Architecture Components for OZIX 128

CHAPTER 11 SYSTEM INSTALLATION AND SYSTEM DIRECTORY LAYOUT 133

11.1 Introduction 133

11.2 Goals 133

11.3 System Directory Layout 134

11.4 System Installation 134

11.5 Application Installations 135

11.6 Internationalization . . . 135
11.7 Related Documents . . . 135

CHAPTER 12 PERFORMANCE ENGINEERING . . . 137

12.1 Abstract . . . 137
12.2 Introduction . . . 137
12.3 OZIX Performance Engineering Goals . . . 137
12.4 OZIX Performance Methodologies . . . 138
12.4.1 Workload Characterization . . . 138
12.5 OZIX Instrumentation and Data Collection . . . 138
12.5.1 Overview . . . 138
12.5.2 Event Detection . . . 139
12.5.3 Sampling . . . 139
12.5.4 OZIX Instrumentation Data Collection Facility . . . 139
12.5.5 OZIX Instrumentation Services . . . 140
12.5.6 OZIX Instrumentation Architecture . . . 140
12.5.6.1 Security Implications . . . 141
12.6 OZIX Performance Characterization Tools . . . 142
12.6.1 System Performance Characterization Tools . . . 142
12.6.2 Component Performance Characterization Tools . . . 142
12.6.2.1 Elapsed Time Analysis . . . 143
12.6.2.2 Execution Profiling . . . 143
12.6.2.3 Coverage . . . 143
12.6.3 Initial Performance Characterization Tools . . . 144
12.6.4 Capacity Planning . . . 144
12.7 OZIX Modeling Strategy - Overview . . . 145
12.7.1 OZIX Modeling Goals . . . 146
12.7.2 OZIX Modeling Strategy . . . 146

CHAPTER 13 SOFTWARE QUALITY AND TESTING STRATEGY . . . 149

13.1 Overview . . . 149
13.1.1 Motivation . . . 149
13.1.2 Software Quality and Testing Strategy Requirements . . . 149
13.1.3 Software Quality and Testing Strategy Goals . . . 150
13.2 Software Quality and Testing Strategy Components . . . 150
13.2.1 System Specifications . . . 150
13.2.1.1 Benefits . . . 151
13.2.2 Code Reviews . . . 151
13.2.2.1 Benefits . . . 152

13.2.3 Unit and Integration Testing 152
13.2.3.1 Benefits 153
13.2.4 Executive Testing 153
13.2.4.1 Benefits 153
13.2.4.2 Nub Routines Test Suites 153
13.2.4.3 Executive Subsystem Tests 154
13.2.5 Application Level Testing 154
13.2.5.1 Benefits 154
13.2.5.2 Commands and Utilities Test Suite 154
13.2.5.3 API Subsystem and Library Test Suite 154
13.2.5.4 DECwindows/Xwindows Test Suite 154
13.2.5.5 AIA Routines Test Suite 155
13.2.6 Base Level Regression Testing 155
13.2.6.1 Benefits 155
13.2.7 Problem Tracking 155
13.2.7.1 Benefits 155
13.2.8 Test System Coverage Analysis 155
13.2.9 Full-time and Stand-alone Usage of the System 156
13.2.10 OZIX System Installation Verification Procedure 156
13.2.10.1 Benefits 156
13.2.11 Field Test 156
13.2.12 Software Documentation Testing 156
13.2.13 Stress/Load Testing and Subsystem Interaction 157
13.2.14 Configuration Testing 157
13.2.15 Network Testing 157
13.2.16 Interoperability Testing 157
13.2.17 System Management Testing 157
13.2.18 Conformance/Compliance Testing 157
13.2.19 Usability Testing 157
13.2.20 B2 Certification 158
13.2.21 Hardware Testing Strategy 158
13.2.21.1 Hardware Testing Overview 158
13.2.21.2 Online Diagnostic Monitor 158
13.2.21.2.1 ODM Architecture 158
13.2.21.2.2 Online Diagnostic Program Architecture 159
13.2.21.3 System Exerciser 159

13.3 Overall Quality and Testing Strategy 160

13.4 Risks 160

13.5 Associated Documents 160

GLOSSARY OF TERMS 161

FIGURES

1 OZIX Operating System 3
2 Overlap of the OZIX User Environments 12
3 Ways Users Access OZIX 15
4 Ways Applications Access OZIX 27
5 Basic System Administration Structure 36
6 Simple Management Application and Object Relationship 38
7 Complex Management Application and Object Relationship 38
8 Modeling Example 39
9 Technical Interoperability 42
10 System Administration Components 43
11 Resource/Fault Management Overview 49
12 Security Ring Levels 62
13 A simple subsystem 88
14 Subsystem Execution Context 90
15 Executor SPC calls 91
16 OZIX Virtual Memory Model 92
17 Logical segment page fault resolution 93
18 Physical memory page reclamation 94
19 Overview of OZIX I/O System 98
20 ABA Storage, Migration, and Archival Domains 101
21 OZIX Mass-storage I/O Interfaces 104
22 Device Driver Components 107
23 Terminal Subsystem and Network Drivers 111
24 Network Architecture Overview 115
25 Internet Network Components 116
26 DECnet/OSI Network Components 119
27 Network Management Components 121
28 Client and Server Stub Interaction 124
29 OZIX System Directory Tree 134
30 The OZIX Instrumentation Architecture 141
31 The Common Collector Facility 141

TABLES

1 DoD Model Labels 70
2 Commercial Model Labels 71
3 Comparison of DoD and simple Commercial Access Control Models 73

Preface

This document provides a technical summary of the OZIX system. For a marketing analysis of the OZIX program, see the *OZIX Vision Document*.

CHAPTER 1

INTRODUCTION

1.1 The OZIX Project

OZIX is an extensible, hardware-independent operating system that integrates modern technologies for distributed systems, fault tolerance, data integrity, and high-performance I/O. OZIX is structured to fully support the process management, file processing, system calls, and communication services defined in the POSIX™, X/OPEN™, OSF™, and ISO standards. The reliability, data integrity, and security requirements of transaction processing applications are also supported as core features in the OZIX system.

The entire complement of OZIX features will be implemented over a series of releases. Such features include system elements, such as database and transaction processing software, compilers, tools, and utilities from other Digital groups, in addition to selected elements from third-party vendors.

Version 1 of OZIX is a high-performance production system targeted at the open systems market. While transaction processing technologies are built into Version 1, these capabilities are not made visible to higher levels of software until later versions. Version 1 is designed to function as a network and compute server in a multivendor, distributed systems network. Version 1 serves this network by providing file services (via NFS™) and compute services (RPC and user applications). Version 1 also includes integrated system and network management features, license management, and CDROM distribution. Portability of applications is provided with application integration architecture (AIA) components such as DECwindows™ client, Compound Document Architecture (CDA), and Concert Multithread Architecture (CMA).

Project Priorities

In order to maintain a productive focus, the goals of the OZIX project are prioritized in the following order:

1. OSF/AES and ULTRIX™ interoperability
2. Seven-days-a-week, Twenty-four-hours-a-day (7x24) Reliability
3. Performance, Robustness
4. B2 Security
5. Internationalization
6. Transaction Processing Primitives
7. IBM® Interoperability
8. Cost
9. Time to Market
10. UNIX® Interoperability

11. Portability
12. VMS™ Interoperability

1.2 Building the Foundation of a Modern Operating System

Hardware and system designs have progressed significantly over the last decade. The fundamental assumptions made when designing operating systems 10 years ago – 1 VUP CPUs and 256 kilobytes of memory—no longer apply to today's systems. These assumptions have made it very difficult to retrofit new concepts into existing operating systems.

Back in 1978, higher performance processors were 1 VUP, 2 megabytes of memory was considered more than enough, demands on disk storage were low, networks were small, local area networks didn't even exist, and no one was concerned about security or internationalization. Today, we have hardware offering 50+ VUPs that can support hundreds of megabytes of memory; users are pushing the limits of disk storage; networks must support thousands of nodes consisting of systems from multiple vendors; there is an urgent demand for security; and the fastest growth in computer sales is taking place in the international markets. Further, customers are requiring standard user interfaces on systems that deliver the highest performance possible.

The OZIX project offers unique opportunities to deliver a system that can keep pace with these rapid improvements in technology and increasing customer needs. Many of these opportunities stem from building a contemporary operating system that is based on modern concepts, along with an understanding of the current and future demands of the production system marketplace. The OZIX system, however, is as much a product of design methodologies as it is of modern technologies. As an operating system built from the ground up, every component is methodically designed and tested to ensure that performance, reliability, security, system management, and internationalization are cleanly integrated into the overall structure of the system.

1.3 Building an Open System

OZIX is a modern operating system platform that integrates the existing and emerging open system standards defined by POSIX, X/OPEN, OSF, and ISO/OSI. Support for standards—such as those defining user interface, data exchange, networking, file access, multi-processing, transaction processing, security, and internationalization—are pervasive throughout the OZIX system. Particular efforts are made to support the applications, networking, and system/network management standards necessary to ensure the success of OZIX in distributed system environments. Conformance testing ensures that the OZIX implementation of these standards conforms to the standards definitions.

1.4 The OZIX Architecture: A Structured Approach

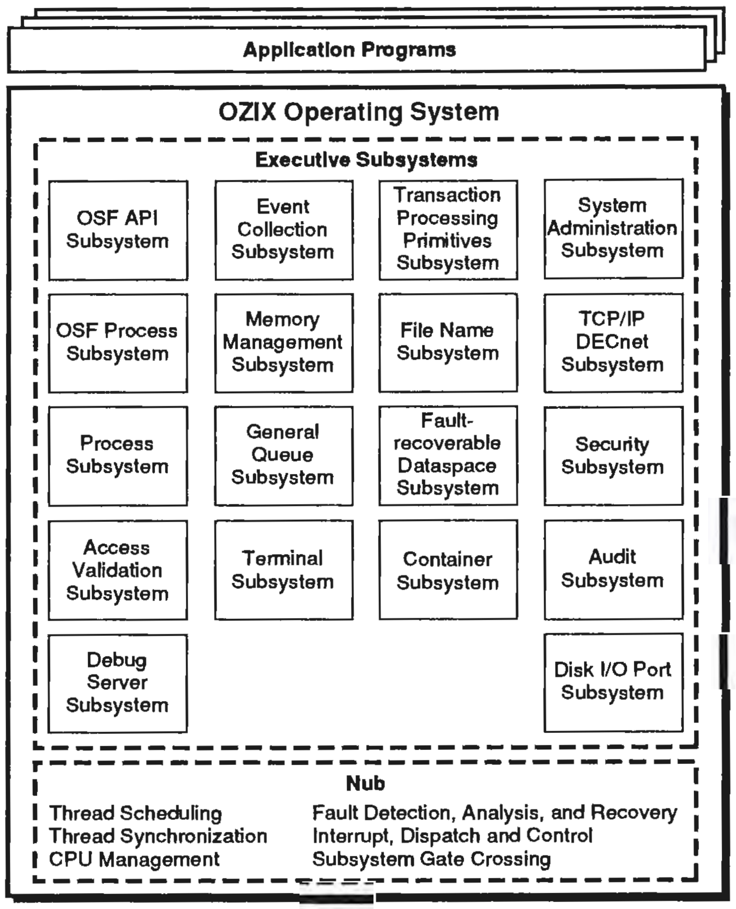
The OZIX architecture defines a highly structured environment within which system components are easily designed and implemented to deliver maximum performance, reliability, flexibility, and security. The primary goal of the OZIX system is to provide a solid foundation for transaction processing (TP) and customized applications. To accomplish this goal, particular emphasis is placed on supporting modern fault recovery and I/O technologies that provide superior availability and integrity of data across a large number of storage devices.

In order to ensure that OZIX remains competitive through the 1990s and beyond, the OZIX system is modular and extensible enough to adapt to new technologies and interface standards as they evolve.

Subsystems

As illustrated in Figure 1, the OZIX architecture defines functional units called *subsystems*. Each subsystem is implemented as part of the executive to support a specific functional area in the OS. Separate subsystems are used, for example, to support application programming interfaces, the file system, memory management, I/O devices, and so on.

Figure 1: OZIX Operating System



Subsystems reduce the complexity of the entire system by breaking it down into functional units. Each subsystem can be understood on an individual basis, without the need to understand all of the other subsystems in the operating system. Such functional division simplifies the task of design, as well as the task of updating the system to accommodate new features.

The Nub

The OZIX system is designed to keep hardware-dependent code to a minimum. Hardware-dependent operations—such as multiprocessor coordination, thread dispatching, condition handling, timer support, and so on—are implemented by a small, low-level component called the *nub*. By restricting all of the processor-dependent code to the nub, it is not necessary to modify the rest of the system when porting OZIX to new hardware.

High Performance

OZIX is designed to make maximum use of multiple-processor, high-performance hardware. The OZIX architecture defines a very efficient multi-threading environment in which threads are created and destroyed quickly and with minimal overhead. OZIX supports a wide range of thread priority levels, including real-time priorities. In order to deliver fast real-time response to high priority requests, threads executing in the executive can be preempted by higher-priority threads. This OZIX multi-threading environment is complemented by OZIX compilers and runtime environment, which feature thread-safe libraries, minimized locking contention, and multi-thread debug capabilities.

The OZIX architecture specifies a highly reliable symmetric multi-processing (SMP) environment by defining a locking strategy that allows data structures to be locked independently. This locking strategy includes lock levels, which enforce a method of locking that ensures deadlock-free operation.

Overall system performance is gained by making optimum use of physical memory. For example, some of the code and data in the executive is pageable. In addition, memory allocation and management routines are provided to support a variety of paging algorithms for increased intelligence in memory paging.

To ensure that the performance goals of OZIX are met, the design and implementation of the system is closely monitored to identify bottlenecks, and simulation methods are developed for testing OZIX performance on various hardware platforms.

Reliable Operation

Integral to the OZIX design is a resource/fault management strategy that provides a wide range of error detection and correction capabilities. This strategy incorporates rule-based failure prediction, error detection, error logging, fault analysis, and system configuration control to predict, detect, and isolate hardware and software errors.

One of the benefits of the OZIX subsystem design is that errors occurring in a subsystem are easily isolated, and are not allowed to propagate to other subsystems. Such architectural firewalls allow subsystem failures to be handled without bringing the whole system down.

Other OZIX features that enhance the reliability of the system include a common logging mechanism, stack-based exception handling, disk shadowing, and the ability to dynamically reconfigure storage devices.

B2 Security

One of the primary goals of OZIX is to provide high levels of system security and integrity. The functional isolation between subsystems make it possible to build a system with levels of security and integrity enforcement that are unobtainable by any other means.

OZIX is designed to obtain a B2 level of certification by the Department of Defense (DoD). The constraints imposed by DoD security policies, however, are not suitable for most commercial applications. OZIX, therefore, supports multiple security and integrity policies to satisfy the needs of both government and commercial environments. Each of these security and integrity policies are enforced with the full strengths of a B2 system.

Maximum security and integrity enforcement has an impact on system performance. Since not all customers in all environments require the full B2-level of enforcement, the system can be tuned using a wide variety of enforcement parameters to provide the appropriate mix of performance, security, and integrity for a given customer environment.

Security is discussed in more detail in the Security chapter.

1.5 User and Application Support

OZIX supports standard user interfaces such as X/Windows and POSIX-compliant shells, commands, and utilities. In addition to these standards, the OZIX executive also incorporates Digital enhancements that integrate internationalization into the system.

Application Programming Interface (API) Subsystems

Application programming interface (API) subsystems serve a key role in implementing interfaces, such as those defined by OSF and POSIX. APIs implement the features that are unique to the various programming interfaces supported by OZIX. (For example, the OSF API implements file descriptors, signals, and so on.) When an application issues a system service call to an API subsystem, the API subsystem either handles the service directly, or translates the service into one or more subsystem procedure calls, which invoke the appropriate subsystems to complete the request.

OZIX is designed to host multiple APIs to support a variety of interface standards. OZIX, Version 1 is to be delivered with an OSF API, which supports POSIX, X/OPEN, and OSF applications.

Shared Libraries

The executive supports shared libraries. The shared library features include:

- Position-independent libraries, so that libraries are not fixed to virtual addresses at library compile time.
- Dynamic symbol resolution allows the symbol resolution to be deferred until actually used.
- Optional load-time symbol resolution allows all symbols to be resolved at image start-up time and prevents unexpected symbol resolution during operation.
- Partial default bindings at link time allows for faster symbol resolution.
- Version and interface type control ensures that compatible entry points are used.
- User and programmer control of library selection eases development and customization.

As the OSF definition evolves, the features listed above will be enhanced to conform to the OSF interface definition.

International Support

A major goal of the OZIX effort is to provide an internationalized computing environment capable of supporting applications that span national, linguistic, and cultural boundaries. Such internationalization support is pervasive throughout OZIX—in libraries, terminal services, the file system, the base system architecture, and the message facility. OZIX supports the characters for most every language through the implementation of the multiple octet character set (MOCS). In MOCS, up to 4 bytes can be used to define each character to support large character sets, such as Japanese Kanji, Chinese Hanzi, and South Korean Hangul and Hanja.

OZIX allows multiple libraries to be built on top of the OSF API subsystem. A standard C library is provided to support existing applications based on 8-bit character sets, as well as libraries utilizing compound strings to support new international applications for world-wide markets.

Internationalization is discussed in more detail in the Internationalization chapter.

1.6 Transaction Processing Primitives

The executive defines services that provide low-level transaction processing support. Components within the executive, such as the file system, and components built on top of the executive, such as a distributed transaction processing system, use these services to achieve atomicity and durability.

The transaction processing services provide the primitives for common logging, generating transaction IDs, recovery of file system and data structures, checkpointing, concurrency control, as well as to begin, end, and abort transactions.

1.7 Process Support

The OZIX executive provides execution support for applications through a process subsystem. The process subsystem is intended to be generic with respect to the API used by the application. For example, the process subsystem can support both a fork/exec model and a create process model, depending on the API being used.

The process subsystem implements process and thread scheduling policy, process signaling, as well as the creation, deletion, and management of processes and threads.

Standard UNIX Interfaces

The following is a list of some of the standard UNIX interfaces supported by the OZIX executive:

- Process Management Services—OZIX supports standard UNIX process management system services, such as fork and exec. The complete list is defined in the Process Management Interfaces section of the *OSF AES Operating System Programming Interfaces System Services Outline, Revision 2.0*. Digital extended COFF is used for the image file format.
- Signal Interfaces— OZIX supports the standard UNIX signal interfaces, such as kill and signal. The complete list is defined in the Signal Interfaces section of the *OSF AES Operating System Programming Interfaces System Services Outline, Revision 2.0*.
- Process Environment Interfaces— OZIX supports the standard UNIX environment interfaces, such as getpid and umask. The complete list is defined in the Signal Interfaces section of the *OSF AES Operating System Programming Interfaces System Services Outline, Revision 2.0*.

Concert Multithread™ Services

The executive supports the Concert Multithread ™ Architecture core services, as defined in Chapter 4 of the *Concert Multithread Architecture, Revision 1.0-2*. This support includes true multithread support in the executive and asynchronous alerts.

Scheduling Features

The executive supports advanced scheduling features to control the scheduling of threads and processes. Possible system options include:

- A fair-share scheduler
- A time share scheduler
- A run to completion with preemption scheduler

1.8 Memory Management Interface

The executive supports the BSD style memory management interface as defined in the Memory Management Interfaces section of the *OSF AES Operating System Programming Interfaces System Services Outline, Revision 2.0.* Mapped files and the data in the file system is automatically kept consistent. This means that, if an application opens a file and issues read/writes while another application has the file mapped, each application can see the changes made by the other.

In addition, memory management supports the following features:

- Maximum sharing of physical memory
- Unlimited number of virtual address spaces (only limited by amount of physical memory)
- Sparse virtual address space
- Multiple page files
- Maximize size of user virtual address space
- Alternate backing store managers and page replacement algorithms
- The ability to use as much physical memory as needed, when physical memory is available
- Fair sharing of physical memory when physical memory is scarce

1.9 I/O Support

The primary goals of the OZIX I/O system are high performance, reliability, and the ability to efficiently manage tens of terabytes of data on hundreds of storage devices. The I/O system supports these goals through the utilization of subsystems, multiple threads, and the Digital *attribute-based allocation* (ABA) architecture.

ABA allows file data to be dynamically mapped to storage devices. This dynamic allocation of data provides the foundation for features such as hierarchical storage management (HSM) and dynamic reconfiguration of storage devices. ABA also provides the basis for disk striping, disk shadowing, volume sets, and logical volumes.

ABA and the OZIX I/O design are discussed further in the I/O System chapter.

1.10 Distributed Computing Environment

OZIX provides a rich distributed computing environment by supporting many defacto network standards. These include the Sun™ Network File System (NFS), a remote procedure call (RPC) facility based on the Apollo® Network Computing System (NCS); distributed naming services via the Berkeley Internet Name Domain (BIND) and Digital's Distributed Name Service (DNS); and authentication services via MIT's Kerberos and Digital Authentication Security Service (DASS).

Network Support

The OZIX network provides the underlying technology for success in a distributed environment. This includes multivendor interoperation via adherence to both defacto and international open standards, a state-of-the-art high performance network implementation, and careful attention to the ability to scale the network to very large configurations.

TCP/IP and DECnet™/OSI

OZIX communication services include both the Internet (TCP/IP) and OSI protocol suites. These protocols provide network interoperability with virtually every major computer vendor. Both are high speed native implementations, accessible to applications through a choice of standardized interfaces, such as Berkeley sockets and the X/OPEN Transport Interface (XTI). Indirect network services, such as remote procedure calls (RPCs) and remote file access, are also provided. All system and network services are managed through a single, common management mechanism.

OZIX adds additional applications and services to provide a wider range of functions to other OZIX, ULTRIX, VMS, and MS/DOS systems. In addition, DECnet/OSI provides gateway access to proprietary IBM SNA networks and CCITT X.25 public data networks.

The OZIX distributed computing environment is discussed in more detail in the Network Implementation Architecture chapter.

1.11 System Administration

OZIX system administration is designed to conform to POSIX, X/OPEN, OSF, and ISO/OSI interface standards, as well as DoD security standards. The OZIX system administration structure is based on object-oriented methodology. All tasks performed by system administration are performed on *manageable objects*, which represent users, devices, files, processes, networks, and so on. As objects, these components are identified by their specific attributes, operations, and events.

Manageable objects are connected through a *management backplane* to each other and to *user presentations*. The management backplane provides a common set of services across interconnected machines. This interconnect is provided by the Enterprise Management Architecture (EMA) and the use of various ISO and IETF network management standards.

A user presentation is a type of *management application* that provides an interface through which system managers manipulate objects on the backplane. Several user presentations are implemented to provide a variety of interfaces for both character-cell and DECwindows terminals. In addition to user presentations, automated management applications are implemented to handle such tasks as system configuration and fault management.

System administration is discussed in more detail in the System Administration chapter.

1.12 Built-in Performance Analysis

The OZIX system is designed to provide built-in performance analysis capabilities through an instrumentation collection subsystem. The instrumentation collection subsystem implements a set of instrumentation services that use collection points in the OZIX code to capture the state of the system at any point in time. Such event collection capabilities provide the information necessary for capacity planning and distributed system debugging. Additional performance information on the OZIX system is provided by performance characterization tools, such as prof, Pixie, and the SA* series of system activity tools.

Performance engineering is discussed further in the Performance Engineering chapter.

1.13 Outline

The following chapters discuss the various OZIX components and implementation strategies in more detail. For complete design information on these topics, refer to the respective functional specifications.

- Chapter 2, User Environment and Application Environment, discusses the OZIX user and application environment. This chapter describes OZIX from the point of view of end users, application programmers, and system administrators.
- Chapter 3, System Administration, describes the overall structure on which the OZIX system administration strategy is based.
- Chapter 4, Resource/Fault Management, discusses how OZIX is to achieve its availability, reliability, configuration, and serviceability goals.
- Chapter 5, Security, discusses the strategies for building the security and integrity into OZIX necessary to achieve a B2 level of certification by the DoD.
- Chapter 6, Internationalization, describes the strategies for building internationalization support into the OZIX system.
- Chapter 7, Base System Architecture, describes the fundamental support mechanisms for the OZIX executive and nub. This chapter discusses how OZIX supports subsystems, manages threads, and implements virtual memory.
- Chapter 8, I/O System, provides an overview of the Digital ABA architecture and discusses the OZIX file system and mass storage designs.
- Chapter 9, Terminal Subsystems, describes how OZIX terminals are supported in the OZIX system.
- Chapter 10, Network Architecture, discusses the OZIX distributed system architecture.
- Chapter 11, System Installation and System directory Layout, describes the OZIX directory structure, as well as the strategies for installing the system software and layered application software.
- Chapter 12, Performance Engineering, discusses the strategies for integrating performance analysis into the basic design of the OZIX system. This chapter also describes how performance instrumentation is to be built into the system for the purposes of performance characterization, system management, capacity planning, transaction processing, and configuration management.
- Chapter 13, Software Quality and Testing Strategy, describes the software testing methodologies used to ensure a quality implementation of the OZIX system.

CHAPTER 2

USER ENVIRONMENT AND APPLICATION ENVIRONMENT

This chapter contains a high-level description of the OZIX V1.0 computing environment as seen by OZIX users. The chapter starts by briefly characterizing OZIX users by type, and then discusses the environment OZIX offers to each of these types of users. For each user environment, the discussion consists of a brief description of the user interfaces and applicable tools as provided on OZIX, any unique OZIX technology used to provide these capabilities, and the major *user-visible* ramifications of this technology.

The purpose of this chapter is to provide a road map to OZIX capabilities from a task- or user-oriented perspective, so as to supplement the component- and subsystem-oriented descriptions that comprise the rest of this technical summary.

2.1 Types of OZIX Users

When discussing the different types of user environments available on OZIX, it is first helpful to divide users into three arbitrary groups.[1]

The first group of users, which we refer to as *end users*, are those users that typically have little contact with anything but the most basic commands and utilities available on OZIX.

The second group of users, which we refer to as *application developers*, are characterized by the software development tools and system routines that they use in addition to the tools used by end users.

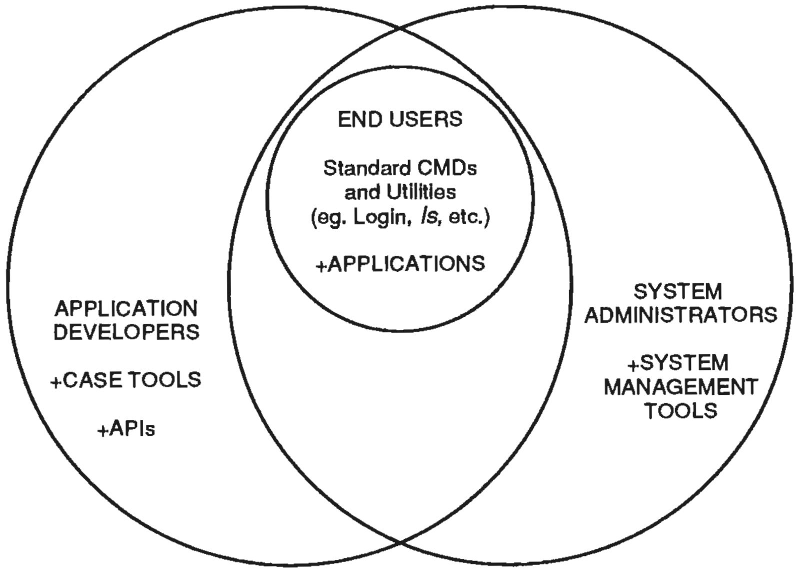
The third group of users, which we refer to as *system administrators*, are characterized by their use of privileged interfaces to OZIX that are used to control the overall function of OZIX.

Figure 2 shows how the user environments of these three groups overlap. This figure is not meant to be rigorous; there are always exceptions to this model. The figure is meant to convey in broad detail that the heart of the OZIX user environment is the end user environment, described in Section 2.3.

In general, all three groups of users use the common commands and utilities contained in the end user environment. Applications developers typically use all of these commands and utilities plus a set of additional utility programs, such as compilers, that are part of the OZIX program development environment. In a somewhat disjoint space, system administrators use many of the end user tools plus a series of system management tools.

[1] This taxonomy of three groups of users is arbitrary. These three groups should be familiar to most readers of this chapter. For a more precise marketing-derived discussion of OZIX users, especially for detailed descriptions of the system administrator user group, see the user model contained in the *OZIX Usability Plan*.

Figure 2: Overlap of the OZIX User Environments



The following three sections describe the main attributes of each type of OZIX user. The rest of this chapter then discusses how all users access OZIX capabilities and goes on to describe each of the OZIX user environments in more detail.

The description of system administrators and their environment contained in this chapter is brief as this information is provided in greater detail in the System Administration chapter and the *OZIX Usability Plan.*

2.1.1 End Users

OZIX end users consist of a broad spectrum of people who use OZIX because their applications run on OZIX. Their key applications run on OZIX because OZIX offers a set of industry standard application program interfaces. End users also benefit from the standard look-and-feel of the OZIX user interface; they bring their experience of working with UNIX, ULTRIX, or OSF/1 systems with them to OZIX.

As a group, OZIX end users are not required to be very computer-literate, but they may be. If they have experience with computers, it is probably based on experience with UNIX, ULTRIX, and OSF/1 systems. Users of turnkey applications may also have experience running on MS-DOS®, VAX/VMS™, or MVS™ systems. In addition to running specific applications, an end user's primary day-to-day tasks include running mail, performing document processing, and performing simple maintenance of their online environment, for example, creating and deleting files. Users of turnkey applications are often aware of only the user environment provided by the application they run.

In general, end users are more interested in getting their job done using the tools available on OZIX, than they are in details of how OZIX works. As a result, they are likely to be unaware of the technical details of many unique OZIX capabilities. Instead, they rely on system administrators to establish default settings for these features for them and then use these features as if they were a "black box." At this level, OZIX enhanced internationalization support and the distributed computing environment is of interest to many end users.

End users use both the character-cell-terminal and workstation interfaces to OZIX; the choice of which type of interface is used is largely based on the economics of per-seat costs of character-cell terminals versus workstations. For users of turnkey applications, the choice of terminal interface is most often determined by the requirements of the applications used.

2.1.2 Application Programmers

OZIX application developers use OZIX because it is based on industry standards. OZIX application developers include "end customers", third-party developers (both ISVs and CMPs), and Digital developers. They develop applications based on OZIX because OZIX offers a set of standard application program interfaces plus Digital added value in the form of a rich set of additional application program interfaces and application development tools. Like end users, application developers also benefit from the standard look-and-feel of the OZIX user interfaces; they bring their experience of working with UNIX, ULTRIX, or OSF/1 systems with them to OZIX.

OZIX application developers are computer-literate; they know how to operate computers at the end-user level and they also know how application programs and operating systems interact. OZIX application developers know how to program and are typically familiar with the C programming language and the application run-time environment of UNIX, ULTRIX, or OSF/1 systems. In addition to performing the same types of tasks that end users perform, application developers design, create, and maintain application programs.

While developing and maintaining their applications, the OZIX application developer's view of the OZIX development environment is similar to the end user's view—using tools as black boxes. Because of their knowledge, application developers are much more demanding about the performance and capabilities of their development environment. Application developers are typically very sophisticated users of shell capabilities; indeed, many application developers use the programmable features of OZIX standard shells as a major supplement to their application code and to customize the development environment. At the level of the development environment, application developers are aware of the presence of OZIX added value and they take advantage of it, for example, by using the additional applications that DECwindows provides. Application developers are also more likely to be aware of capabilities of the OZIX security environment and take advantage of it than are end users.

While designing and upgrading their applications, the OZIX application developer's view of OZIX switches to the level of the application program interface. Application developers make tradeoffs in performance, capability, maintainability, standards-compliance, and so on as an integral part of their job. Application developers take advantage of the presence of added value in OZIX, subject to the tradeoffs mentioned above. For example, the increased internationalization support in OZIX available to applications is of value to application developers.

Application developers use both the character-cell-terminal and workstation interfaces to OZIX; the preferred interface is the workstation interface because of the flexibility that multiple simultaneous windows allow. OZIX application developers are typically very well-versed in the advantages of distributed computing and take advantage of this support from OZIX in both their applications and as part of the OZIX development environment.

2.1.3 System Administrators

The group of users known as OZIX system administrators encompasses a variety of system management and control roles:

- Account Administrator
- Auditor
- Operator
- Security Administrator
- System Programmer
- Network Administrator

Some of the many tasks performed by system administrators include: installing and maintaining the OZIX system and layered products, creating and maintaining user accounts, performing backup and other media management, controlling resources, batch, print, and network control, and controlling the security and integrity features of OZIX,

Because the system administrator role is so broad, it is hard to describe them in general terms. At one end of the spectrum operators are very similar to privileged end users; they use largely fixed operator interfaces to perform a limited range of operations. In this role, system administrators are like turnkey application users. At the other end of the spectrum system programmers are very similar to privileged application developers; they create system-level applications using the same basic environment that application developers use.

OZIX system administrators are computer-literate; they know how to operate computers at a system level. They typically understand the types of resources they control and the implications (in terms of system performance, security, and throughput) of their actions. OZIX added value for system administrators comes in improving the ease-of-use of system administration tools, reducing the level of computer literacy required.

OZIX system administrators are aware of the presence of OZIX added value and the methods used to control it. OZIX' system administration focus is based on a sophisticated window-based user interface that is an obvious improvement over the system management capabilities of other UNIX, ULTRIX, and OSF/1 systems. Control of OZIX' enhanced security and integrity features are of great interest to them, as well. As was mentioned in Section 2.1.1, system administrators are often called upon to establish a usable black-box environment for unsophisticated end users, for example, by establishing the correct internationalization parameters for users.

OZIX system administrators use both the character-cell-terminal and workstation interfaces to OZIX; the preferred interface is the workstation interface because of the existence of OZIX' enhanced system administration support utilities, which require a workstation to run.

For a more detailed description of system administrators see the *OZIX Usability Plan* and the System Administration chapter.

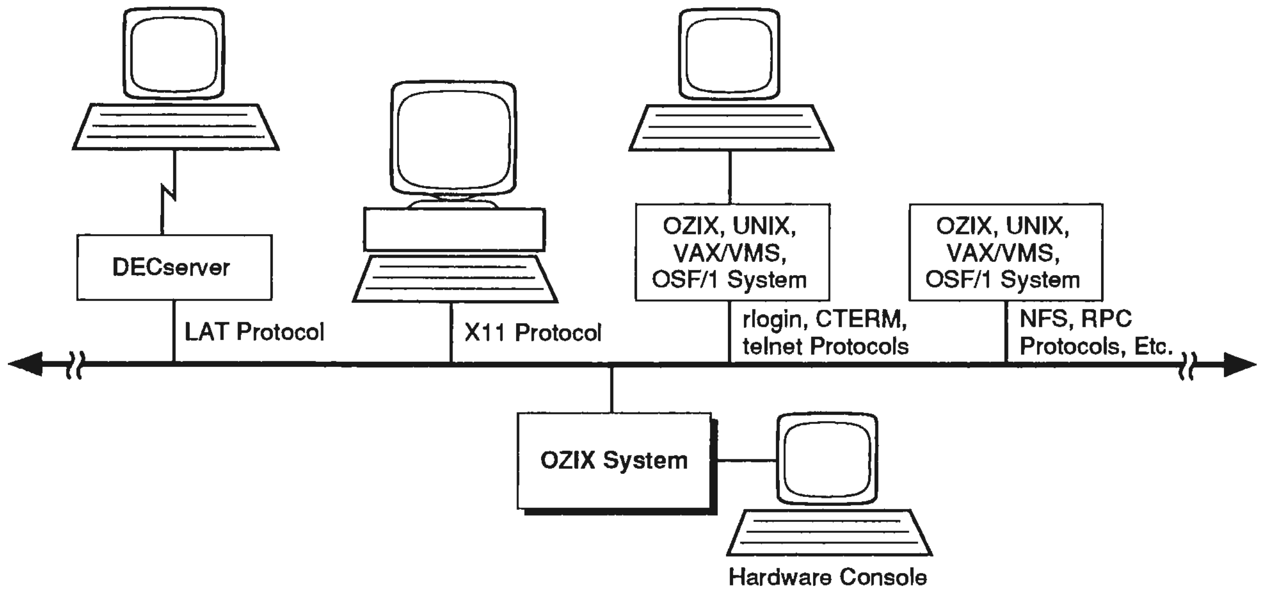
2.2 Ways Users Access OZIX

Like other implementations of OSF/1, users can access an OZIX system by:

- Interactively logging in to OZIX from character-cell terminals or bitmapped terminals/workstations
- Executing applications from a remote system
- Executing applications using the OZIX batch system
- Accessing data that resides on an OZIX system from a remote system

Figure 3 shows a sample networked computing configuration containing an OZIX system, and it shows several examples of the methods for accessing OZIX capabilities. The following sections describe the ways users access OZIX.

Figure 3: Ways Users Access OZIX


DECserver
LAT Protocol
X11 Protocol
OZIX, UNIX, VAX/VMS, OSF/1 System
rlogin, CTERM, telnet Protocols
OZIX, UNIX, VAX/VMS, OSF/1 System
NFS, RPC Protocols, Etc.
OZIX System
Hardware Console

2.2.1 Logging In to OZIX

Most users access OZIX via the interactive interface, logging in to OZIX using character-cell terminals or bitmapped terminals/workstations (hereafter referred to simply as workstations).

There are four major ways to log in to OZIX:

1. Locally on a character-cell terminal connected to the system console terminal interface.

   The user initiates the access process in a hardware-specific manner, typically by pressing BREAK on the console terminal.

2. Through a terminal connected to a terminal server on the network.

   The terminal server is typically a LAT server device such as a DECserver™.

   The user initiates the access process by using the LAT server CONNECT command, which logically connects the terminal to the OZIX login processing software.

3. Remotely from a terminal on another system in the network.

   There are several possible remote login methods:

   — The *rlogin* terminal protocol, which can be used from an OZIX, ULTRIX, OSF/1, or other UNIX-style system.

     The user initiates the access process by using the *rlogin* command, which logically connects the terminal to the OZIX login processing software.

   — The *telnet* terminal protocol, which can be used from OZIX, ULTRIX, OSF/1, other UNIX-style systems, or from *telnet* terminal servers, known as "PADs".

     The user initiates the access process by using the *telnet* command, which logically connects the terminal to the OZIX login processing software.

   — The CTERM terminal protocol, which can be used from an OZIX, ULTRIX, or VAX/VMS systems.

The user initiates the access process by using the *dlogin* (on OZIX and ULTRIX) or SET HOST (on VAX/VMS) commands.

4. From an X11™-based workstation connected to an OZIX system via the network.

   There are several ways to initiate the access process:

   - The user issues an *rsh* command (described in Section 2.2.2) to invoke an X11-based terminal emulator running on OZIX (for example, xterm or dxterm), logically connected to the user's workstation.
   - The user issues an *rsh* command to invoke a visual shell running on OZIX, logically connected to the user's workstation.

The commands and utilities supplied with OZIX are described in more detail in Section 2.3.1. Except for a few noted exceptions, all OZIX commands and utilities support access using the character-cell-terminal interface.

2.2.2 Executing Applications From a Remote System

In addition to logging in, another primary interactive interface to OZIX is via the remote shell capability—the *rsh* command.

Users on remote OZIX, ULTRIX, UNIX, and OSF/1 systems can use the *rsh* command to invoke applications, commands, and utilities on an OZIX system.

The OZIX implementation of *rsh* functionality (both incoming and outgoing) is similar to that provided with UNIX, ULTRIX, and OSF/1 systems. For incoming requests, the OZIX Internet daemon receives *rsh* protocol messages sent to the local OZIX system from a remote system. It invokes the *rsh* daemon to execute the specified shell commands, and it performs the pipefitting necessary to ensure that the local *rsh* daemon can communicate with the remote *rsh* code.

2.2.3 Executing Applications Using the Batch System

Users may also access OZIX by submitting jobs to the OZIX batch facility, using the *at* and *batch* commands, as specified by X/OPEN. These commands are issued directly on OZIX, or equivalently, may be executed on OZIX from a remote system through the use of the *rsh* command.

The *at* and *batch* commands buffer shell commands to be executed via the batch facility. Depending on the command used to submit the batch job, processing is performed slightly differently.

Batch jobs submitted using the *at* command are queued by execution time in a data file. The data file is periodically examined by a daemon scheduled via an entry in */etc/crontab*. When a job is found that is due to execute, the daemon forks and executes a shell to process the batch shell script. The daemon performs pipefitting so that, unless redirected by the the batch shell script, the contents of *stdin* and *stdout* are saved and mailed to the submitter upon job completion.

Batch jobs submitted using the *batch* command are queued for immediate execution. When ready for execution, batch jobs are processed in the same way as jobs submitted using the *at* command.

2.2.4 Accessing Data on OZIX From a Remote System

Users on remote systems can access data on OZIX by using:

- Network File Services (NFS) support

  OZIX NFS support exports portions of the OZIX filesystem to NFS clients (for example, UNIX, ULTRIX, OZIX, OSF/1, Macintosh® and MS-DOS systems). To access OZIX data using NFS, the remote user merely references the desired filename (using *cat*, for example), using a path name local to their system; the access is largely transparent to the remote user and remote software making the access.

- DECnet Data Access Protocol (DAP) support

  How remote users access OZIX data using DECnet support depends on the type of operating system running on the remote system.

  For users on remote VAX/VMS systems, the user can access OZIX files by using RMS-compatible file specifications that reference the OZIX system's DECnet node name and specify the file. As with NFS support, these file specifications are usable by most VAX/VMS software (for example, COPY) to access the OZIX file.

  For users on remote ULTRIX systems, the user can access OZIX files by using the DECnet support utilities (for example, *dcp, dcat, and dls*). These utilities allow the remote users to copy the OZIX files to their system for local processing. Unlike NFS support and DECnet support on VAX/VMS, only these support utilities are able to access files on OZIX; all other commands, utilities, and applications are not capable of directly accessing OZIX data.

- ISO File Transfer Access Method (FTAM) support

  How remote users access OZIX data using FTAM support depends on the type of operating system running on the remote system.

  For users on ULTRIX systems (V4.0 and later) and OZIX systems, the *ftam* or *cp* commands can be used to copy data from an OZIX system to the remote system for processing there.

  For users on remote VAX/VMS systems, the COPY command can be used to copy data from an OZIX system to the VAX/VMS system for processing there.

- File Transfer Protocol support

  Remote OZIX, OSF/1, ULTRIX, and UNIX users can copy files from OZIX to their remote system for local processing using the *ftp* command.

- Berkeley file transfer support

  Remote OZIX, OSF/1, ULTRIX, and UNIX users can copy files from OZIX to their remote system for local processing using the *rcp* command.

2.3 The OZIX End User Environment

As shown in Figure 2, the end user environment is the "common ground" for all OZIX users. This common environment is based heavily on standardized software.

The following sections describe the major aspects of the OZIX end user environment:

- Section 2.3.1 describes the standard commands and utilities provided with OZIX that are typically used by end users.
- Section 2.3.2 describes the shells provided with OZIX.

- Section 2.3.3 describes the major features of the OZIX character-cell-terminal user interface.
- Section 2.3.4 describes the major features of the OZIX workstation user interface.
- Section 2.3.5 and Section 2.3.6 describe the user's view of the added value provided with OZIX' enhanced security and internationalization support.

2.3.1 Standard Commands and Utilities

OZIX provides a wealth of commands and utilities whose user interface and function comply with the following formal industry standards and *de facto* industry standards:

- X/OPEN Portability Guide, XSI Commands and Utilities (Issue 3) —an industry standard
- POSIX 1003.2, Shell and Application Utility Interface for Computer Operating System Environments (Draft 8) – an industry standard
- OSF Operating System Component (OSC) Functional Outline: Application Environment Specification (Revision 1.0) – a *de facto* industry standard

In addition, other commands and utilities are provided with OZIX to allow for migration of users from VAX/ULTRIX to OZIX.

End users use a limited set of the total commands and utilities provided with OZIX to perform the following types of basic tasks on OZIX:

- Maintaining files and directories

  Commands used for these types of tasks include: *cat, rm, ls, mkdir, cd, cp, mv, more, view, grep*, and so on.
- Communicating with other users and systems

  Commands used for these types of tasks include: *mail, rcp, talk*, and so on.
- Document processing

  Commands used for these types of tasks include: *vi, spell, nroff, troff, tbl*, and so on.
- Printing documents

  Commands used for these types of tasks include: *lpq, lprm*, and so on.
- Obtaining information

  Commands used for these types of tasks include: *man, who, ps, date*, and so on.

For more information about standard commands and utilities provided on OZIX, see the *OZIX Standard Commands Functional Specification* and the *OZIX ULTRIX-32 Compatibility Commands Functional Specification*. Together, these documents list the names of all standard commands and utilities provided on OZIX.

2.3.2 OZIX Shells

Both character-cell terminal users and workstation users can interface to OZIX using the following shells:

- Bourne Shell—a standard shell
- C Shell—a *de facto* standard shell

These shells provide a simple, line-oriented command interface and function identically to other implementations as provided on UNIX, ULTRIX, and OSF/1 systems. The standardized look and feel of these shells reinforces to users that OZIX is based on standards.

Like other OSF/1-compliant open systems, OZIX allows users to choose a default shell. This shell preference is stored as part of the per-user data managed by the system administration components. Users may also change shells dynamically, as is done on other OSF/1-compliant open systems.

Workstation users may also use a window-based visual shell known as the ULTRIX User Executive. The ULTRIX User Executive provides an icon-based visual user interface to OZIX based on a paradigm of direct manipulation of files to achieve a particular task. The user interface to UUE is common between OZIX, ULTRIX, and VAX/VMS. UUE is not available on non-Digital platforms, such as generic UNIX systems; UUE is part of the added value Digital provides as part of DECwindows.

Users invoke the ULTRIX User Executive in the same way they invoke other window-based applications (see Section 2.3.4.1). Through a customization menu provided by UUE, users can modify and extend various aspects of the function of UUE, including modifying functional defaults (such as default directory used by UUE) and cosmetic defaults (such as the size and color of the windows used by UUE).

In terms of its implementation on OZIX, UUE is just another window-based application that makes calls to the DECwindows API.

2.3.3 Character-Cell-Terminal User Interface

Access to OZIX via the character-cell-terminal interface provides users with a command-line interface that is a superset of the standard command-line interfaces provided on OSF/1, ULTRIX, and UNIX systems. This interface is characterized by the use of a shell that presents a programmable interface to all OSF-specified standard commands and utilities, plus some additional Digital-standard and OZIX-specific commands and utilities.

The character-cell-terminal interface to OZIX is implemented in various components, depending on the type of access involved:

- Access via the console terminal

  The OZIX console port driver and terminal class driver provide the basic software support for this style of access.

- Access via a LAT-based terminal

  The OZIX LAT port driver and terminal class driver coupled to other network software that provides the software support for this style of access.

- Access via the *rlogin*, *telnet*, SET HOST, or *dlogin* commands

  The OZIX PTY port driver and terminal class driver coupled to other network software that provides the software support for this style of access.

2.3.4 Workstation User Interface

Using a character-cell-terminal emulator, workstation users can access OZIX via the character-cell-terminal interface. In addition, workstations users can run X11, OSF/Motif™, and DECwindows applications on OZIX to present a window-style user interface back to their workstation. These applications are hereafter referred to simply as window-based applications.

A set of standard window-based applications is provided with OZIX, including:

- A character-cell-terminal emulator
- A window-based login capability and session manager

OZIX adds value to the standard X11-based and OSF/Motif-based user interfaces in many ways by supporting DECwindows. DECwindows added value includes:

- Additional Digital-supplied window-based applications:
  - DECwindows MAIL
  - ULTRIX User Executive
  - DECwindows OOTB applications, such as the calendar manager.
  - Digital's DECchart, DECwrite, and DECdecision applications
- Enhanced internationalization support:
  - Enhanced support for bidirectional and multilingual text, implemented using Digital's compound string technology. For more information about compound strings, see the Internationalization chapter.
  - Enhanced support for localization of interfaces, implemented using Digital's user interface language (UIL) technology.
- Enhanced application support, implemented using Digital's DECwindows Toolkit enhancements, resulting in a higher degree of consistency in the look-and-feel of application human interfaces.

2.3.4.1 Invoking Window-based Applications

Users can run window-based applications on OZIX with the human interface directed to an X11-compatible workstation on the network. The user starts the application on the OZIX system and specifies the network address of the target workstation.

OZIX provides several ways to start up window-based applications, all similar to mechanisms used on UNIX, ULTRIX, and OSF/1 systems today. These are the primary methods:

- From a workstation, the user can log in to OZIX and issue a command to start up the application. The user logs in using the mechanisms described in Section 2.2.1. The network address of the target workstation can be specified explicitly on the command line, or implicitly through the DISPLAY environment variable.
- From a local login session on a workstation, the user can invoke the *rsh* command to remotely issue a command on OZIX to start up an application. The network address of the target workstation is often explicitly specified as part of the commands specified via the *rsh* command.
- On the OZIX system, the user can submit a batch job (via *at* or *batch*) that starts a window-based application. Alternatively, a system administrator can place an entry in */etc/crontab* to start a window-based application for the user. These techniques can be used to cause the execution of a shell script that periodically starts or stops the execution of window-based applications. The network address of the target workstation is often explicitly specified as part of the commands specified via the *at* command, *batch* command, or the entry in */etc/crontab*.

These are just the most common techniques, others are possible. Most of these techniques rely on using either the standard interactive login support (as described in Section 2.2.1), remote shell support (as described in Section 2.2.2), or batch system support (as described in Section 2.2.3) in some combination to invoke the application.

2.3.5 The End User View of OZIX Security

Security on OZIX can be viewed as a continuum that ranges from systems with minimal security support enabled to systems with full security support enabled. As a result, the end user view of OZIX security varies from system to system, based on the security policy in effect on the system.

On OZIX systems with minimal support enabled, most end users perceive the security aspects of running on OZIX as being largely indistinguishable from other OSF/1-compliant systems:

- The OZIX login dialogue and authentication process (based on user name and password) is very similar to that provided on other OSF/1 systems.
- The concepts of "processes" (on OZIX, executors) having a user identification, group identification, and other basic security information are the same on OZIX as on other OSF/1 systems.
- File protection (rwx) and security characterizations (owner, group, and world) and the commands to manipulate protections are the same on OZIX as on other OSF/1 systems. For example, OZIX users can use the standard *chown* and *chmod* commands to modify file ownership and protection.

Security support provided with OZIX gives system administrators a great deal of flexibility in tailoring the security profile of each individual OZIX system. They can implement their local security policy using the following types of controls:

- Access Control Lists (ACLs)
- Mandatory Access Control (MAC)
- Integrity Access Control (IAC)
- Level control of network access
- Control over access to “raw” devices

In addition, OZIX security support allows the security administrator to change the basic security model in effect on the system. For example, some OZIX systems may use a security model based on the Bell and LaPadula model; other systems may create their own customized security model. There are a variety of other tunable security features provided with OZIX.

As more security features are enabled on OZIX systems, the end user is likely to perceive this change as the direct result of increasing access limitations – information access and exchange becomes subject to more control and restrictions. For many OZIX systems, the OZIX security system works quietly “behind the scenes” and is only visible to end users when an access attempt fails. The access error is reported to most end users via the standard OSF/1 interfaces, for example, via a standard error report of "permission denied’ (EACCES) from a standard command or utility.

At the highest level of security support, OZIX systems conform to the National Computer Security Center (NCSC) B2 level of security support.

The following two sections describe two major user-visible aspects of OZIX security: access control lists and the Trusted Computing Base Shell (TCB Shell).

For more information about the concepts and capabilities of the OZIX security system, see the Security chapter, the *OZIX Security Support Component Functional Specification*, and the *OZIX Access Validation Component Functional Specification*.

2.3.5.1 The End User View of Access Control Lists

OZIX systems support the use of ACLs to label files (actually, dataspaces), devices, and shared memory segments with access control information (identifiers plus access type allowed). On many systems, ACLs are established and maintained by system administrators and the use of ACLs is largely invisible to OZIX end users. This type of environment is likely to be found at installations where compatibility with the UNIX end user environment is paramount.

For other systems, and for other types of users such as application developers and system administrators, explicit manipulation of ACLs by users will be more prevalent. Generally, explicit ACL manipulation by these types of users is performed to supplement and enhance the capabilities provided by the basic OSF/1 security model. For example, ACLs can be used to partition the user community with a finer granularity than "owner", "group", and "world."

2.3.5.2 The End User View of the TCB Shell

End users also interact with the Trusted Computing Base Shell (TCB Shell), a user interface to the Access Validation Subsystem (AVSS) and Security Support Subsystem (SSSS). The interaction with these components of the OZIX security system occurs either implicitly or explicitly.

Users implicitly interact with the TCB Shell when they log in to an OZIX system. The TCB Shell intercepts notification of the intention of the user to log in to OZIX. On terminals connected via LAT, the notification of the intention to log in is the result of the CONNECT command. For the console terminal, it is the result of the user pressing BREAK. For other types of access, including *rsh* access, a shell command is used to invoke the TCB Shell.

As part of login processing, the TCB Shell works in concert with AVSS to validate the login attempt. The user is prompted for a login access type \assumed to be logically equivalent to "default/secure-access1/secure-access2, etc."\. If the default access type is selected, requesting a normal OSF/1 login access, the user is prompted for user name and password and logs in as if on any other OSF/1 system. If any access type is selected other than the default, the TCB Shell and AVSS work in concert to perform a more sophisticated login procedure. Examples of these non-default login procedures might involve the use of additional authentication mechanisms such as "smart-cards" or biometrics.

Users explicitly interact with the TCB Shell after they are logged in to OZIX, as the result of pressing BREAK on their terminal. This usage of the TCB Shell is to allow a user to modify security-related aspects of their login session (by creating a new executor with the desired security-related attributes), thereby subjectively "changing their privilege level" within the security universe established by the site security manager. An example of this would be to change the secrecy level of their executor from "TOP-SECRET" to "SECRET".

After pressing BREAK on a logged in terminal, the user is logically disconnected from the terminal session based on the executor in effect at the time. This executor is held "in limbo" until it is either destroyed or resumed as the result of user input to the TCB Shell. The user is then logically connected to the TCB Shell, which offers a menu-based interface with the following capabilities:

- Show Access Class

  Selecting this menu choice causes the current values of the following four security parameters (collectively referred to as access class) to be displayed on the user's terminal:

  — Secrecy Level—for example, "TOP-SECRET"

  — Secrecy Compartments—for example, "US-CIA, NATO"

  — Integrity Level—for example, "SYSTEM"

  — Integrity Compartments—for example, "GENERAL, PRODUCTION, TEST"

For more information on the meaning of these terms and an overview of the OZIX security system, see the Security chapter.

- Set Access Class

  Selecting this menu choice causes the TCB Shell to present another sub-menu, allowing the user to specify values for the access class parameters listed above. Note that the secrecy and integrity labels are specified using a single-valued parameter, chosen from a list of values established by the site security officer. Secrecy and integrity compartment labels are specified as a list of one or more values established by the site security officer. The TCB Shell displays possible labels for each of the access class parameters; it only displays those labels that the user is authorized to use.

- Exit the TCB Shell

  If menu choices made by the user modified the security access class of the executor, the old executor (the one that existed prior to invoking the TCB Shell) is destroyed[1] and the new executor is created with the new access class, running the user's default shell. The user is logically connected to the new executor; the effect is as if the user logged in again.

  Otherwise, if no modification of the access class of the executor was requested (for example, if access class information was only displayed and not modified), the user is logically reconnected with the old executor and continues the interrupted session as if the interaction with the TCB Shell had not occurred.

2.3.6 The End User View of OZIX Internationalization Facilities

The enhanced internationalization support available in OZIX is visible as part of the end user environment in three ways:

- Enhanced support for non-ASCII text and international data provided by the standard commands and utilities.
- Enhanced support for non-ASCII terminals and locale-specific terminal input processing provided by the terminal class driver.
- The mechanisms used to control these enhanced capabilities.

The following sections discuss each of these topics in turn.

2.3.6.1 Internationalization and Standard Commands and Utilities

OZIX adds value to the standard commands and utilities mainly in the area of increased internationalization support. With respect to internationalization support, there are three types of OZIX commands and utilities:

- Commands and utilities that are functionally equivalent to the existing standard-compliant commands and utilities available on UNIX, ULTRIX, and OSF/1 systems.

[1] It is destroyed because it is not possible to change the security access class of an executor. Although this chapter talks loosely of changing or modifying the security access profile of an executor, and that is largely the effect the user sees when using the TCB Shell, the actual mechanism is to create a new executor and destroy the old one. It is undecided if the TCB Shell will also allow "parking" of the old executors instead of destroying them; this would allow users to have access to several executors, accessible one at a time.

The OZIX versions of these commands and utilities perform equivalently with other implementations on UNIX, ULTRIX, and OSF/1 systems. How well each command and utility supports execution in a non-English, non-ASCII environment depends on which standard the command is compliant with; X/OPEN XPG3-compliant software performs the best with respect to internationalization support.

In general,these commands and utilities may not process non-ASCII input correctly. They may not be able to support locale-specific processing of dates, currency, collation order, bidirectional text, multiple languages, and so on.

- OZIX versions of selected commands and utilities that have been made "8-bit clean."

  The OZIX versions of these commands are modified so that they will not arbitrarily remove the high-order bit of each input byte they process. Generally, this improves the ability of these commands and utilities to minimally process non-ASCII text without error. Like the unmodified commands and utilities mentioned in the previous category, these commands may still be severely limited in their support of processing international data.

- OZIX versions of selected commands and utilities that have been fully internationalized.

  The OZIX versions of these commands and utilities use Digital added value in the form of Digital's extensions to X/OPEN's Natural Language Support (NLS) and use OZIX' compound string API support to support processing multibyte, multilingual, and bidirectional text data. These commands and utilities also allow a user to specify preferences for locale-specific processing of dates, currency, collation order, bidirectional text, multiple languages, and so on.

2.3.6.2 Internationalization and Terminal Driver Support

OZIX also adds value to the OSF-specified character-cell-terminal interface by adding extensive support for character-cell terminals that use character code sets other than ASCII or the Digital Multinational Character Set (MCS). In addition to supporting ASCII- and MCS-based terminals, OZIX fully supports the use of many non-ASCII/non-MCS terminals such as Digital's VT382 terminal (used to support Asian markets).

The level of internationalized terminal support available on any single terminal connected to an OZIX system also depends on the level of hardware support for transmitting/receiving arbitrary character codes over that hardware. For example, the console terminal interfaces available on many OZIX hardware platforms are often limited in terms of supporting non-ASCII/non-MCS character set data. Accordingly, internationalized terminal capabilities are typically reduced when accessing OZIX via the console terminal on these hardware platforms.

The OZIX terminal class driver offers customizable locale-specific terminal processing support for input preprocessing and output postprocessing. An example of input preprocessing would be support for Japanese-language input preprocessing; this capability allows for user-customized input of Japanese using several, equivalent representations of a word. An example of output postprocessing would be in the conversion of an internal processing codeset representation of output, to a terminal-specific codeset. The terminal driver allows for the dynamic insertion of these locale-specific processing modules.

For more information about the OZIX terminal drivers, see the I/O System chapter and the *OZIX Terminal Subsystem Structural Overview*.

2.3.6.3 How End Users Control Internationalization Support

Most aspects of OZIX internationalization support are affected through the use of the *setenv* command. The *setenv* command is used to specify environment values used for modifying the behavior of the NLS and messaging components of OZIX. This method for controlling internationalization support is specified as part of the X/OPEN Portability Guide.

To control the internationalization support provided by the OZIX terminal driver, a slightly different approach is used. A human user would typically be unaware of the mechanism used to specify the desired international terminal support, as it is most commonly set as part of the login process, based on the user's profile. Users can dynamically change the terminal class driver's processing behavior, for example to change input preprocessing, as in Japanese terminal support.

2.4 The OZIX Application Development Environment

The following sections describe the two halves of the OZIX application development environment:

- Section 2.4.1 describes the OZIX program development environment: the application developer's view of OZIX.
- Section 2.4.2 describes the OZIX run-time execution environment: the application's view of OZIX.

2.4.1 OZIX Programming Tools

OZIX programming tools are based on the use of standard commands and utilities (described in Section 2.4.1.1) as specified by X/OPEN, OSF, and POSIX. As such, the look and feel of OZIX programming tools are mostly identical to those provided on other UNIX, ULTRIX, and OSF/1 systems. Section 2.4.1.2 describes some of the Digital added value provided by the OZIX programming tools.

2.4.1.1 Standard Commands and Utilities

OZIX application developers create and maintain applications using a variety of programming languages such as:

- Ada™
- BASIC
- C
- C++
- COBOL
- FORTRAN
- Pascal

Application developers use all of the commands and utilities used by end users (see Section 2.3.1) plus additional commands and utilities to perform the following types of tasks on OZIX:

- Create and compile source code

  Commands used for these types of tasks include: *cc, cb, lint, lex, yacc, gencat*, and so on.
- Link and debug applications

  Commands used for these types of tasks include: *ld, dbx*, and so on.
- Analyze application performance

Commands used for these types of tasks include: *pixie, prof*, and so on.

- Manage the program development process

  Commands used for these types of tasks include: *make, sccs*, and so on.

- Create utility programs and sophisticated shell scripts

  Commands used for these types of tasks include: *grep, find, awk, sed, echo, tee*, and so on.

- Distribute applications using remote procedure call (RPC) technology

  The *nidl* command is used to invoke the Network Interface Description Language (NIDL) compiler used to convert RPC entry point definitions into C source code.

2.4.1.2 Digital Added Value Commands and Utilities

For the most part, the presence of Digital added value in the OZIX program development environment is the direct result of added value in the application run-time environment (see Section 2.4.2.4). OZIX adds value to the standard program development environment by:

- Enhancing the linker (*ld*) to create shared libraries. Several new *ld* command options have been added to support this. For more information, see the *OZIX Linker (ld) Functional Specification*.
- Adding support for building and loading subsystems.
- Adding support for building secure applications.
- Adding support for generating enhanced message catalogs. This support allows for the storage of message text in compound string format.

The look and feel of these additional commands and utilities (or extensions to existing standard-specified commands and utilities) is patterned after the UNIX-style command line interface style specified by X/OPEN, POSIX 1003.2, and OSF.

For workstation users, DECwindows added value includes:

- The User Interface Language (UIL) Compiler

The look and feel of the DECwindows tools is consistent with the *DECwindows Style Guide*.

2.4.1.3 Third-Party Tools

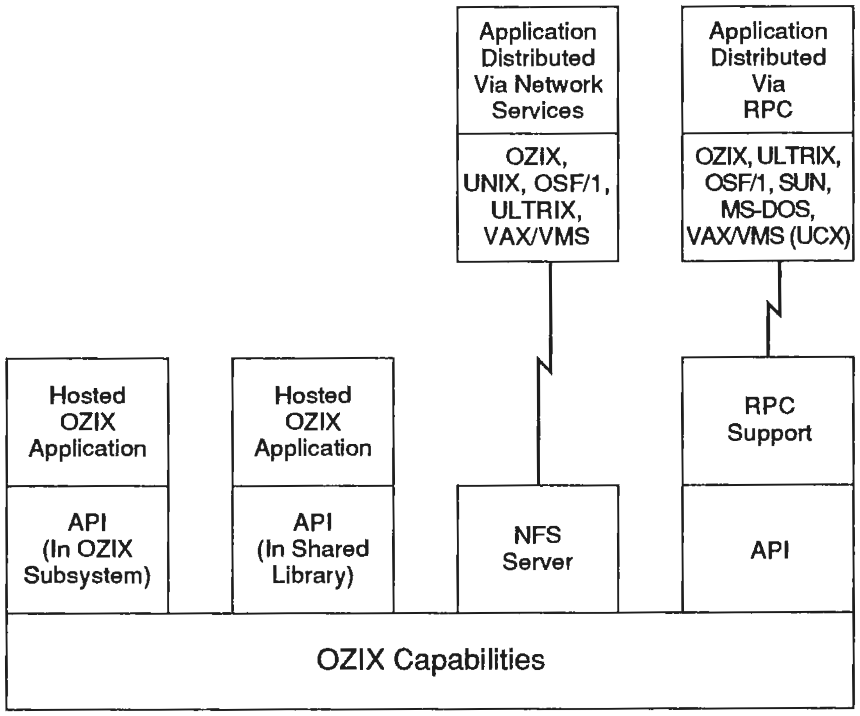
OZIX will support the execution of the INGRES™ relational database product, including Structured Query Language (SQL) support.

For more information about third-party tools on OZIX, see the *OZIX Applications Plan*.

2.4.2 The Application Execution Environment

Figure 4 shows a sample OZIX computing environment in which four applications are running.

Figure 4: Ways Applications Access OZIX


Application Distributed Via Network Services
OZIX, UNIX, OSF/1, ULTRIX, VAX/VMS
Application Distributed Via RPC
OZIX, ULTRIX, OSF/1, SUN, MS-DOS, VAX/VMS (UCX)
Hosted OZIX Application
API (In OZIX Subsystem)
Hosted OZIX Application
API (In Shared Library)
NFS Server
RPC Support
API
OZIX Capabilities

The figure gives four examples of the three basic ways applications access OZIX capabilities:

- Executing directly on OZIX

  The leftmost pair of applications shows the most common way applications access OZIX capabilities—by executing directly on OZIX and invoking documented application program interfaces. OZIX supports all of the common mechanisms for packaging API components: source files (.h files), object files (.o files), object module archive files (.a files), and so on.

  The leftmost two applications in Figure 4 also show two additional interface packaging techniques that OZIX adds to the list above. One application references an API implemented using an OZIX subsystem; the other application references an API implemented using a shared library. Section 2.4.2.4 describes the unique features of OZIX subsystems and shared libraries.

- Accessing OZIX via a network server interface[1]

  The third example in Figure 4 shows an application that accesses OZIX via a network server interface. Examples of this type of application access supported on OZIX are Network File System (NFS) server support and DECnet Data Access Protocol (DAP) support.

- Invoking OZIX capabilities via RPC

  The fourth example in Figure 4 shows an application that accesses OZIX via a specialized form of network access support—remote procedure call.

[1] The orientation of this chapter is on other systems accessing OZIX systems. It should not be overlooked that OZIX systems also originate many types of network requests.

OZIX includes support for Digital's DECrpc V1.0. DECrpc is Digital's version of HP/Apollo Computer's *de facto* industry standard RPC architecture incorporated in their Network Computing System (NCS). RPC support on OZIX allows applications to be distributed between OZIX and a variety of systems that support the NCS protocol:

— OZIX systems

— ULTRIX (V4.0 and later) systems

— VAX/VMS (with VMS/ULTRIX Connection V1.4 and later) systems

— Apollo Computer systems

— Sun Computer systems

— MS-DOS systems

— and others

Briefly, to distribute a portion of an application the developer creates a description of the procedure interfaces in that portion of the application. The partitioning of the application is based on procedure interface boundaries. These procedure interface descriptions are expressed using the NIDL language. The NIDL interface descriptions are then compiled into C source files using the NIDL compiler supplied on OZIX.[2]

The resulting C source files are compiled and are called stubs. They are linked with the application program and also with the portion of the application being distributed. The stubs provide the interface to the RPC run-time support library. When the application transfers control to one of the distributed procedures, it actually transfers control to one of the stub routines described above. This stub routine invokes the RPC run-time library to locate the distributed portion of the application on the network and to connect to the stub routines contained in it. The remote stub routines convert RPC protocol messages into the actions required to execute the distributed procedure code and return any outputs back to the original caller.

For more information about RPC on OZIX, see the Network Implementation Architecture chapter and the *OZIX Remote Procedure Call Functional Specification.*

2.4.2.1 Standard Application Program Interfaces

OZIX provides a set of standard entry points that are a superset of OSF/1 systems, based on the following overlapping set of documents:

- OSF Operating System Component Functional Outline— Open Software Foundation™
- OSF Application Environment Specification—Operating System Component— System Service Outline, Revision 2.0
- ANSI X3J11 for Programming Language C
- POSIX 1003.1 Portable Operating System Interface for Computer Environments
- X/OPEN Portability Guide, Issue 3, Volume 2, System Interfaces and Headers
- X/OPEN Portability Guide, Issue 3, Volume 7, Network Services
- Federal Information Processing Standard (FIPS) Publication 151

[2] This example assumes that the RPC is originating on an OZIX system. For a more rigorous and complete description of the process of using RPC, see the *OZIX Remote Procedure Call Functional Specification.*

The binary code for these APIs is provided on OZIX in two forms: as object module archive files and also as shared libraries. By default, users link to the shared library versions of these APIs; to link the object module archive versions into an application, users must explicitly mention the archive file when linking. Header files (.h files) are also provided for those components that require them. File names of object module archive files and header files are as specified by standards. File names of shared library files will be specified as part of the detailed design. It is assumed that the file names of shared library files will differ from the object module archive files only in file type (for example, *.sl*).

The following lists some of the standard routines provided on OZIX:

- Standard ANSI C routines
- FORTRAN routines
- Other language run-time support routines
- Math routines
- Termcap routines
- Plot routines
- Curses and X11-based curses library routines
- Primitive DBMS routines
- BSD Socket and X/open Transport Interface (XTI) routines
- GKS routines
- PHIGS routines
- X11 Toolkit and widget support routines
- OSF/Motif support routines

Even though the functional behavior of these routines is specified by standards, the OZIX implementations of *selected* routines adds value by making these routines "8-bit clean" (see Section 2.3.6.1 for a description of the benefits of 8-bit clean code) or by making these selected routines safe for use in multithreaded applications ("thread safe").

For more information, including a full listing of the system calls and library routines supplied with OZIX, see the *OZIX Standard C Libraries Functional Specification* and the *OZIX Miscellaneous ULTRIX Libraries Functional Specification.*

2.4.2.2 Digital Added-Value Application Program Interfaces

OZIX adds value to the application program run-time environment by offering:

- Improved functionality over other OSF/1 systems
- Improved robustness, integrity, and security over other OSF/1 systems
- Improved compatibility or interoperability with other systems

The following lists examples of OZIX added value in the application run-time environment:

- DECwindows Toolkit routines— more functional than X11, OSF/Motif routines
- Compound Document Architecture DDIF support routines – more functional than other OSF/1 offerings

- Compound Document Architecture DDIF Viewer routines – more functional than other OSF/1 offerings
- Concert Multithread Architecture – compatibility with other Application Integration Architecture (AIA) platforms
- DECprint™ interface— more functional than other OSF/1 offerings
- DECnet support routines— interoperability with ULTRIX and VAX/VMS
- Application Portability Architecture routines – compatibility with other AIA platforms
- Compound String support routines – more functional than other OSF/1 offerings
- Compound String-compatible entry points to standard interfaces – more functional than other OSF/1 offerings
- Online Diagnostic Monitor (ODM) interface routines – more functional than other OSF/1 offerings
- OZIX extensions to messaging – more functional than other OSF/1 offerings
- OZIX Security Support Component routines (see Section 2.4.2.4) – more functional than other OSF/1 offerings
- Logging and error recovery system routines – more functional than other OSF/1 offerings

2.4.2.3 Third-Party Application Program Interfaces

Application interfaces to Ingres and for SQL support are bundled with OZIX.

For more information about third-party applications on OZIX, see the *OZIX Applications Plan.*

2.4.2.4 OZIX Added-Value Application Program Interface Technology

This section describes the use and benefits of four OZIX-specific technologies used to enhance the application run-time environment:

- API shared library technology

  The OZIX linker (*ld*) allows for linking several object modules into an aggregate called a shared library. OZIX shared library technology is functionally similar to VAX/VMS shareable image technology. Subsequent link operations using a shared library cause the linker to resolve external references contained in the shared library without instantiating a copy of the referenced code into the resultant application. Instead of instantiating a copy of the referenced object modules, the linker instantiates a small amount of data used by the image loader to dynamically resolve all references to the shared library. This effectively decouples the application image from any libraries it references, with the following benefits:

  — Reduced application image size

    The descriptors placed in the application image are much smaller than the complete object modules they refer to.

  — Reduced system memory usage

    Shared libraries are loaded once and shared among all applications that reference them.

  — Enhanced upwards compatibility and ease of packaging

Shared libraries are dynamically loaded; updates can potentially be performed while OZIX is running. Shared libraries allow for updates to APIs to be packaged in such a way that relinking of existing applications is not required when a new version of an API is installed. Shared libraries allow for version control; shared libraries can be marked so as to allow for run-time detection of mismatch between an application and shared libraries it references.

The linker uses three hierarchically ordered search lists to locate shared library files during link operations. These lists correspond to system-level, group-level, and user-level defaults. During linking, the user-level default list is searched first, followed by the group- and system-level lists in turn. This set of default lists means that most OZIX application developers need not do anything explicit to link to the shared library versions of OZIX APIs. Application developers can create their own shared libraries and connect them into the default linker shared library search path.

For more information about shared libraries, including restrictions for their use, see the *OZIX Linker (ld) Functional Specification* and the *OZIX Ld Utilities Functional Specification*.

- Internationalization support technology

  OZIX provides several enhancements to standard OSF internationalization technology, such as NLS. Most of these enhancements are based on two technologies: "8-bit clean" interfaces and compound strings.

  The benefits of 8-bit clean application program interfaces are logically similar to the benefits of 8-bit clean commands and utilities (as described in Section 2.3.6.1). Applications that process string data without removing the high-order bit of each byte of string data and that do not make arbitrary codeset- or locale-specific processing decisions, can use 8-bit clean APIs to successfully process much internationalized text data, including European and Japanese data. 8-bit clean applications are still typically limited in their support of multilingual and bidirectional text, however.

  Compound strings are a data abstraction that allows for the storage and processing of multilingual, bidirectional internationalized text containing multiple codesets. On OZIX, compound strings are based on using a universal four-octet codeset, the multiple-octet character set (MOCS). To use compound string technology, applications must be written or converted to use the various compound string support routines provided with OZIX.

  The impact of using compound strings on the application developer is that they must abandon the simple "string=array of bytes, manipulated at will" programming paradigm in favor of a "compound string=totally opaque datatype, manipulated only by APIs" programming paradigm. The benefits for making this leap are that all known languages, including Hebrew and Arabic, can be correctly processed by any compound-string-based application.

  Uses of compound string technology in the application run-time environment include:

  — Compound string utility library

    This library provides entry points for manipulating compound string data in an internationalized fashion. Operations include: substring extraction, assignment, concatenation, collating, and so on.

  — Compound-string-compatible versions of standard library entry points

    These entry points are functionally equivalent to many system calls, but allow for the manipulation of string data using compound strings, instead of NUL-terminated strings. For example, the *cs_printf* procedure is used to format and output compound strings to *stdout*.

  — Compound string data stored in OZIX message catalogs

This capability allows an application to store, retrieve, and format message text stored in compound string format.

For more information about compound string support and internationalization support on OZIX, see the Internationalization chapter.

- Security support technology

  OZIX enhanced security technology is available to OZIX application developers (including third-party application developers) in several forms by:

  - Providing APIs used for the development of secure applications.

    The application interface to the security system is provided by the OZIX Security Support component routines. These routines perform auditing, quota management, and covert channel control. See the *OZIX Security Support Component Functional Specification* for more information.
  - Allowing third-parties to install new access control models – these include security, integrity, discretionary, trustedness, or entirely new models.
  - Allowing third-parties to install new access control classes.
  - Allowing third-parties to install new security authenticators.
- Common language calling standard technology

  OZIX provides common interlanguage calling technology with the following features:

  - Language interoperability
  - Parallel multithread application execution
  - Reliable exception handling, including multilanguage uplevel-GOTO support

2.5 OZIX System Administrator Environment

In addition to the commands and utilities used by end users, OZIX system administrators use specialized system administration tools to control the operation of OZIX systems. Section 2.5.1 describes the system administrator environment provided by the standard system administrator commands and utilities. Section 2.5.2 describes the added value OZIX provides in the system administrator environment.

For more information about OZIX system administration, see the *OZIX System Administration Overview* and the System Administration chapter.

2.5.1 Standard System Administration Tools

The system administrator environment as specified by OSF is based on a series of individual commands and utilities. These commands and utilities each have a single function and support a limited character-cell-terminal user interface. Examples of these commands and utilities are: *adduser, removeuser, df, ps*, and *kill*.

Although these commands and utilities are implemented using the OZIX management backplane (described in Section 2.5.2), their function and user interface conform to standards, with a few OZIX-specific extensions.

A complete list of the standard system administration commands and utilities may be found in the *OZIX System Administration Functional Specification*, *OZIX Standard Commands Functional Specification*, and *OZIX Compatibility Commands Functional Specification*.

2.5.2 OZIX Added Value

The heart of OZIX added value in the system administrator environment is the object-oriented technology of OZIX system administration, which embodies Enterprise Management Architecture (EMA) concepts.

OZIX system administration provides a conceptual framework in which system administration activities across a wide variety of tasks, including user access management, security management, storage management, network management, resource management, and configuration management can all be described in a consistent fashion. The OZIX system administration paradigm is based on a universe in which management applications use the management backplane to perform management operations on manageable objects. As a result of this simple paradigm, the OZIX system administrator environment provides a single, unified management control center for all of the types of tasks listed above.

The OZIX management backplane provides a common software interconnect allowing centralized system administration across a variety of networked systems, including OZIX and ULTRIX systems. As it is based on ISO/OSI standards, the OZIX management backplane can be extended to manage systems provided by other vendors.

For more information about the OZIX management backplane, see the System Administration chapter and the *OZIX Management Services Functional Specification*.

In addition to the standard system administration commands and utilities described in Section 2.5.1, OZIX provides three added value system management tools:

- The Management Command Language (MCL) management tool

  This tool provides a character-cell-terminal user interface to manage any manageable object on OZIX in an object-independent fashion. MCL is an extension to the DECnet Network Control Language. MCL is invoked using the *mcl* command and is used for managing objects not managed by the standard commands and utilities and for use with command scripts. This tool is also used in a B2 secure environment to provide a certifiable, trusted path from the system administrator to the managed object.

- The generic object manager

  This tool provides a window-based workstation user interface to an object-independent general management tool. Invoked by the *mcl -d* command, this tool is functionally equivalent to the MCL tool described above. The only difference between the two tools is in the user interface.

- The storage manager

  This tool provides a window-based workstation user interface to a specialized tool used to configure and control aspects of OZIX relating to the Attribute Based Allocation (ABA) storage management technology. This tool presents a sophisticated user interface to manipulate containers, devices, files, dataspaces and other ABA-related objects.

The character-cell-terminal-style tools described above all employ a standard verb/subject-type of command line interface. The window-based tools use the visual paradigm of direct manipulation of objects that is common among window-based applications.

Another added value tool provided in the OZIX system administration environment is the Online Diagnostic Monitor (ODM). ODM provides a character-cell-terminal interface used to control the execution of online diagnostic programs. ODM is also used to control the execution of the Online System Exerciser (OX). For more information about ODM and OX, see the Software Quality and Testing Strategy chapter, the *Online Diagnostic Monitor Functional Specification*, and the *Online System Exerciser Functional Specification*.

2.6 Summary

OZIX is based on standards: user interface standards, command and utility program standards, and application program interface standards. Within this framework of standards, OZIX provides several unique enhancements: an improved user interface, flexible security support, improved internationalization support, improved system administration capabilities, and basic technology improvements over other OSF/1 systems.

This combination of industry standards and Digital added value result in a truly flexible and powerful system suitable for a wide spectrum of applications and users.

CHAPTER 3

SYSTEM ADMINISTRATION

3.1 Introduction

This chapter describes the system adminstration of OZIX Version 1.0 . This chapter introduces the reader to the problems with system administration and the goals of OZIX system administration. This chapter explains the three major components of OZIX system administration: the manageable object, the management backplane, and the user presentation. Interoperability is discussed, followed by a list of system administration components and references to related documents.

3.2 Problem Summary

System administration historically has been developed in an as-needed fashion. When new functions have been invented, new methods and tools have also been invented to manage those functions, often as an afterthought. Any relationship that the new function has with the rest of the system is often left to the system administrator to resolve. An overall framework for system administration has never been defined.

The state of system administration today is poor:

- Production systems are extremely complex and expensive to manage
- Non-production systems are frequently severely limited in their management functionality
- Both production and non-production systems suffer from a preponderance of ad-hoc facilities and interfaces and lack support for distributed and network environments
- No two heterogeneous systems can be managed as a whole.

Solving the system administration problem is a non-trivial task. It requires multiple generations of product offerings as stated in the *OS/SB Enterprise Management Approach: An Overview*. The solution also requires coordinated activity across a large number of groups both internal and external to Digital. This chapter describes the overall structure or framework for system administration in OZIX Version 1.0 . This structure is a foundation on which solutions can be built and is a first step in the evolution of system administration.

3.3 Goals

The goals of OZIX system administration Version 1.0 are as follows:

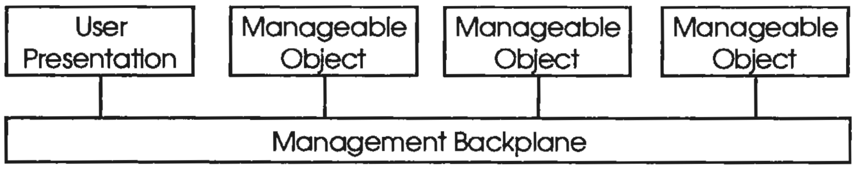
- Provide the first step to managing a network of systems. System administration is not limited to a single node.
- Reduce the cost of system administration of OZIX systems participating in a local area network (LAN).

- Integrate management tasks. Management is performed based on administrative task instead of operating system feature.
- Provide for consistent management. Both philosophy and interfaces for management are consistent across all objects to manage.
- Conform to the emerging Enterprise Management Architecture (EMA). EMA, using the International Organisation for Standardisation's (ISO) Open System Interconnection (OSI), is a definition of a framework for management of heterogeneous, multi-vendor, distributed computing environments and the communications facilities that link them together. It is a Digital corporate-wide architecture.
- Provide adherence and compatibility with Open Software Foundation (OSF), Institute Of Electrical and Electronics Engineers, Inc.'s (IEEE) Standard Portable Operating System Interface for Computer Environments (POSIX), and the X/OPEN Company Limited's X/OPEN Portability Guide. The *OZIX Product and Market Requirements Document* contains the priority order and conflict resolution guidelines.
- Provide management of a production-quality, mass storage system.
- Provide administration of an OZIX system that meets Department of Defense security class B2 rating.

3.4 OZIX System Administration Structure

The structure of system administration is based on object-oriented methodology. The basic building block of system administration is the manageable object. *Manageable objects* are interconnected through a *management backplane* to each other and to *user presentations*. This basic concept is illustrated in Figure 5.

Figure 5: Basic System Administration Structure



The following sections describe each of these in more detail.

3.4.1 Manageable Object

Object-oriented design is a method in which the design is based more on the objects than on the operations performed. The invocation of operations on objects is consistent across all objects. There is much text written on the subject of object-oriented design and programming, and the reader is assumed to have a basic knowledge of the concepts. Because there is no standard terminology used in the object oriented design world, this chapter defines the terminology used in OZIX system administration.

The fundamental building block of OZIX system administration is the manageable object. All tasks required by system administration are performed using the basic structure called a manageable object. Manageable objects are registered with the management backplane. Note that the term manageable object does not relate to the term object as used by the OZIX base system. When referring to manageable objects this chapter simply uses the term object.

For the sake of clarity, a file will be used as an example of an object through this discussion. An object consists of the following basic concepts:

- *Object class*

  Object class is the definition of the object. It is the abstract data structure or template for the object. The object class definition consist of attributes, operations, and events. The object class is analogous to a record or data structure definition in a programming language. An object class has a name which uniquely identifies it from all other object classes.

  A file object class is defined as follows:

Class Name	Attributes	Operations	Events
File	Name	Create	Modified
	Size	Delete	Access Failure
	Creation Date	Set	
	Owner	Get	
	Protection	Backup	

- *Object instance*

  The object instance refers to the actual object. It is sometimes referred to as the instantiation of the object class. The object instance is analogous to the actual record in a data file. An object instance has a name which uniquely identifies it from all other object instances within its object class.

  A file object instance might be as follows:

Attribute	Value
Name	myfile.dat
Size	15
Creation Date	30 October 1989
Owner	JohnDoe

Protection	MaryPublic=read
	JohnPrivate=write

- *Attributes*

  Attributes are internal state variables of the object, which are made visible across the management interface. The object class defines these attributes. Examples of attributes are identifiers, status, counters, and characteristics. In an object instance, each attribute takes on a value and may be examined and modified by an operation on that object instance. In the file object example above, the object class defines the attributes for the file. The object instance of the file object then has values associated with those attributes.

- *Operations*

  Operations are performed on object instances. The object class defines the operations. An operation is initiated outside of the object on which the operation is performed. For the purpose of system administration, operations are performed on object instances, including the *create* operation (the *create* operation is sometimes considered an operation on the object class; however, for consistency, system administration considers it an operation on the instance. This is more of a theoretical difference, as some code somewhere creates an instance of an object).

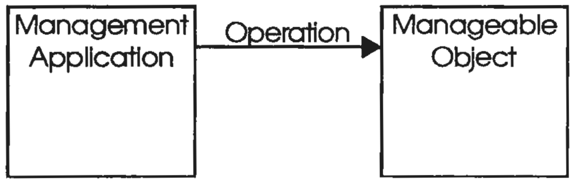
- *Events*

  Objects report events. Events are asynchronous occurrences of normal and abnormal conditions within an object and are triggered by a state transition within the object. For the file object example, a possible event would be an access failure. Access failures are detected when a program tries to open files for which the program does not have access. If access failure occurs, not only would the program be returned a failure status, but the design of the file object may dictate that an event is also reported. The file object does not care who receives the event, only that it must generate the event. In this case, a security monitor may receive the event.

Management operations are performed on objects. This implies that something can request an operation on an object. The requester of an operation is referred to as the *management application*. The relationship between the object and the management application is illustrated in Figures 6 and 7.

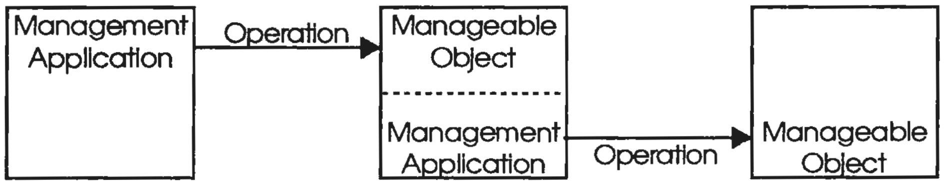
Figure 6 illustrates a simple management application and object relationship. The management application initiates an operation. The object receives the operation and acts upon it.

Figure 6: Simple Management Application and Object Relationship



The roles of management application and object are not static. A management application can in turn be a manageable object if the application requires management. Figure 7 illustrates a more complex set of relationships.

Figure 7: Complex Management Application and Object Relationship



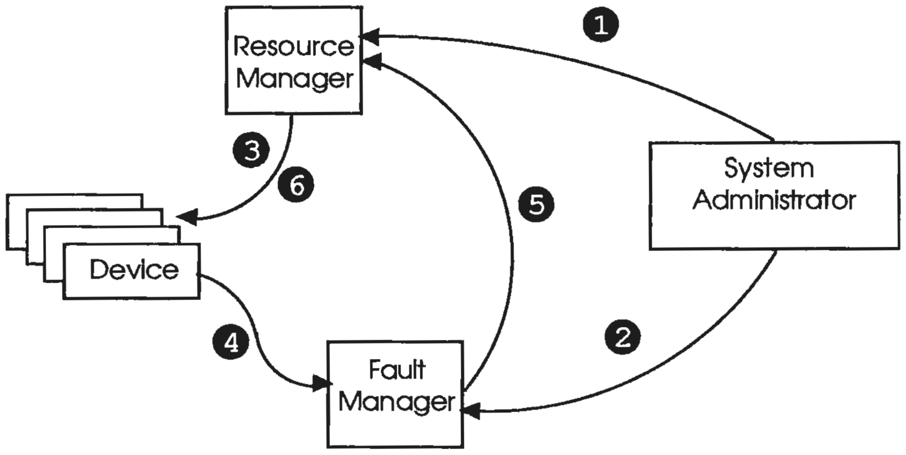
To demonstrate this model, the following objects can be used:

- Devices - These objects represent the devices attached to the system. They have attributes such as on/off line that may be modified with the *set* operation. Device objects also report events when they have errors.
- Fault Manager - This is a management application that monitors the number of soft errors on devices. The fault manager decides, based on error frequency, type, and trends, when a device is failing. It is also an object that is managed. It has attributes that may be modified with a *set* operation, for example, acceptable error thresholds.
- Resource Manager - This is a management application that takes devices online and offline. It is responsible for migrating devices in and out of shadow sets. It is also an object that is managed and has attributes that are modified with a *set* operation. Such attributes include a list of spare devices and the number of replicated devices required in a shadow set.

- System Administrator - The System Administrator uses a management application which is a user presentation. This management application allows the system administrator to communicate to the other objects on the system.

Figure 8 illustrates the objects and management applications described above and some of the relationships and operations that take place. Note that this example is not necessarily the specific model being used in the OZIX product for device fault management.

Figure 8: Modeling Example



❶ The system administrator, using the system administrator user interface (management application), issues the *set* operation on the resource manager (manageable object) to establish the number of replicated devices in each shadow set and to create a list of spare devices.

❷ The system administrator, also using the system administrator user interface (management application), issues the *set* operation on the fault manager (manageable object) to establish error thresholds that the administrator feels his or her computing environment can tolerate.

❸ As a result of ❶, the resource manager establishes the system's resource configuration. This is done by the resource manager (management application) issuing the *set* operation on the device objects (manageable objects) to bring devices online.

❹ As the system runs, the device object (manageable object) reports errors as events. The device object is not concerned with who receives the events; this detail is handled by the management backplane (explained in Section 3.4.2, Management Backplane). In this example, the fault manager (management application) has requested that it receive all error events from the device objects. As the device object reports events, the fault manager receives them.

❺ Using the threshold parameters established by the system administrator in ❷, the fault manager determines that a device is passed the tolerable level of soft errors. The fault manager (manageable object) decides that a device should be removed and reports an event stating the decision. The fault manager is unaware of who receives the event. The receiver might be an automated resource manager (management application) that automatically replaces the device. Alternatively, the receiver might be an operator console (management application), in which case device replacement is a manual operation. The former is the case for this example.

❻ After being notified that a device should no longer be used, the resource manager makes a determination regarding whether the device should be replaced with one from the spare list. If so, the resource manager (management application) issues the *set* operation to the new device (manageable object) to bring the device online as a member of the shadow set, migrates the data from the old device to the new, and then removes the old device from the shadow set.

Modeling the system in this manner promotes a modular design of system administration. Objects and management applications can be replaced without modification of other objects and management applications on the system. For example, if a resource manager is developed that provides a different replacement algorithm, the resource manager can be replaced on the system without modification to the device, fault management, and even possibly the user interface. The same strategy used to create a piece of code using modules, subroutines, libraries, and layering techniques applies to creating objects and management applications.

3.4.2 Management Backplane

The management backplane provides the management services that interconnect objects with each other and with management applications. Some of the management applications are user presentations. The management backplane provides the following services:

- Operation dispatching

  The primary function of the management backplane is to provide object operation dispatching. All management applications request operations on object instances by making the request through the backplane. The backplane is then responsible for passing the operation to the appropriate code that implements the operation on the designated object. This dispatcher is referred to as the *agent*.

- Event services

  Objects report events to the management backplane. The management backplane receives the events through an *event dispatcher* and forwards the event to the appropriate *event sink*. In order to receive events, management applications interested in specific events subscribe to the appropriate event sink in the management backplane.

- Class definitions repository

  The management backplane provides a repository of object class definitions. An object class definition includes those items described in Section 3.4.1, Manageable Object. This allows other objects and management applications to query the management backplane for a description of an object class. Management applications need not have hard-coded details about an object class, but instead, can dynamically retrieve the definition of the object class. This repository is referred to as the Management Information Repository (MIR).

The management backplane also provides the above services across interconnected machines. This implies that a set of machines has a single, interconnected backplane. This interconnect is provided through such architectures as EMA and the use of ISO/OSI network management standards. One of the many benefits this provides is that a user presentation on one machine will be able to manage objects on another machine.

A higher-level architecture for the management backplane can be found in the *OS/SB Enterprise Management Approach: An Overview*.

3.4.3 User Presentation

The user presentation is the system administrator's window to the management of the system. It is a type of management application, and therefore connects to the management backplane.

The decision making that occurs at the user presentation application is not in the application itself, but in the system administrator using the interface. The intelligence that is built into the user presentation is only to make the interface more user-friendly and aid the system administrator in decision-making.

In the user presentation, the system administrator is the decision maker or at least the observer. Compare this to other management applications such as a resource management application and a fault management application. In these cases, the decisions about management are made by the management application based on some rules and parameters.

For OZIX system administration version 1.0 , the following user presentation management applications are provided:

- Standards-compliant command line user presentation

  In order to work in an open system environment, system administration provides a command line which adheres to industry standards. The standards adhered to and the priority order used to resolve conflicts are listed in the *OZIX Product and Market Requirements Document*. The exact commands are specified in the *OZIX System Administration Functional Specification*, *OZIX OSF Commands*, and *OZIX BSD Commands*.

- Management Command Language (MCL) user presentation

  The MCL user presentation is an extension of the DECnet Network Control Language (NCL) utility. MCL allows management access to any manageable object (assuming normal security checks). MCL has no object-dependent, built-in information or intelligence. It only has knowledge of the object class definition of each object: attributes, operations, and events. MCL is used for managing objects not managed by standards compliant command-line user presentations or the DECwindows user presentations described below. Additionally, MCL is used for debugging, maintenance purposes, or command script processing.

  For certification under B2, a trusted path must be provided from the system administrator to the object being administered when performing security-related management. For this reason, a trusted user presentation is provided, which is a subset of the MCL interface.

- DECwindows user presentations

  DECwindows user presentations are the key to added-value in system administration for the OZIX product. There are two DECwindows presentations provided with OZIX: a generic object manager and a specialized manager.

  The generic object manager is a DECwindows equivalent to MCL. This user presentation manages any object in a generic fashion. As with MCL, no object dependent knowledge is in this DECwindows user presentation.

  True added-value comes with providing specialized, task-based, DECwindows user presentation. The specialized DECwindows user presentation for OZIX Version 1.0 system administration highlights the emerging Attribute Based Allocation (ABA) storage management technology.

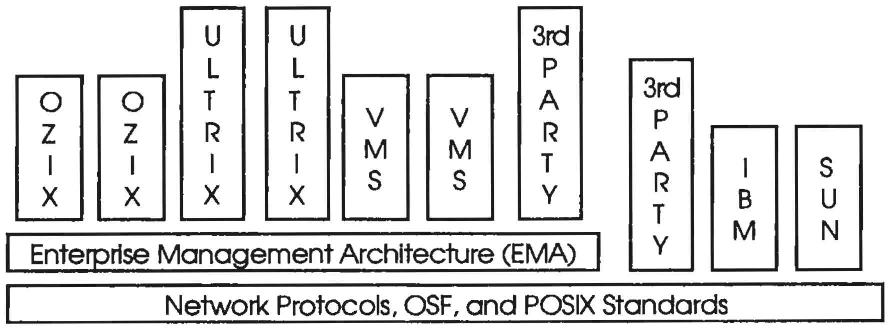
3.5 Interoperability

If OZIX system administration creates the "best" system administration system and ignores interoperability with other systems and applications, system administration has failed. OZIX system administration must interoperate as defined below:

- Interoperability within the system, or better put, *intra*operability. The system must be administrated as a system. Failure to achieve intraoperability precludes interoperability in any form. Intraoperability is achieved by cooperation and review among components of OZIX during the specification and design of the system.
- Interoperability with other systems in a network. This includes both OZIX systems, other Digital systems, including ULTRIX and VMS, OSF-compliant systems, and other vendor ISO-compliant operating systems.

  Interoperability with other systems is achieved on two planes: technical and conceptual. Figure 9 shows the OZIX strategy for achieving technical interoperability with other systems. Important tools in achieving interoperability are Digital's EMA, ISO/OSI management standards and protocols, and adherence to other standards such as OSF and POSIX.

Figure 9: Technical Interoperability


OZIX
OZIX
ULTRIX
ULTRIX
VMS
VMS
3rd PARTY
3rd PARTY
IBM
SUN
Enterprise Management Architecture (EMA)
Network Protocols, OSF, and POSIX Standards

  Conceptual interoperability of system administration is achieved largely through common object definitions. For example, the basic concept of files is similar on all machines. Common object definition also allows for variances such as different file name syntax.

- Interoperability with applications that build on top OZIX. These include Digital-supplied and third-party-supplied applications. Easy incorporation of these applications into the management framework allows the system administrator to manage these applications as part of the system using the same methodology.

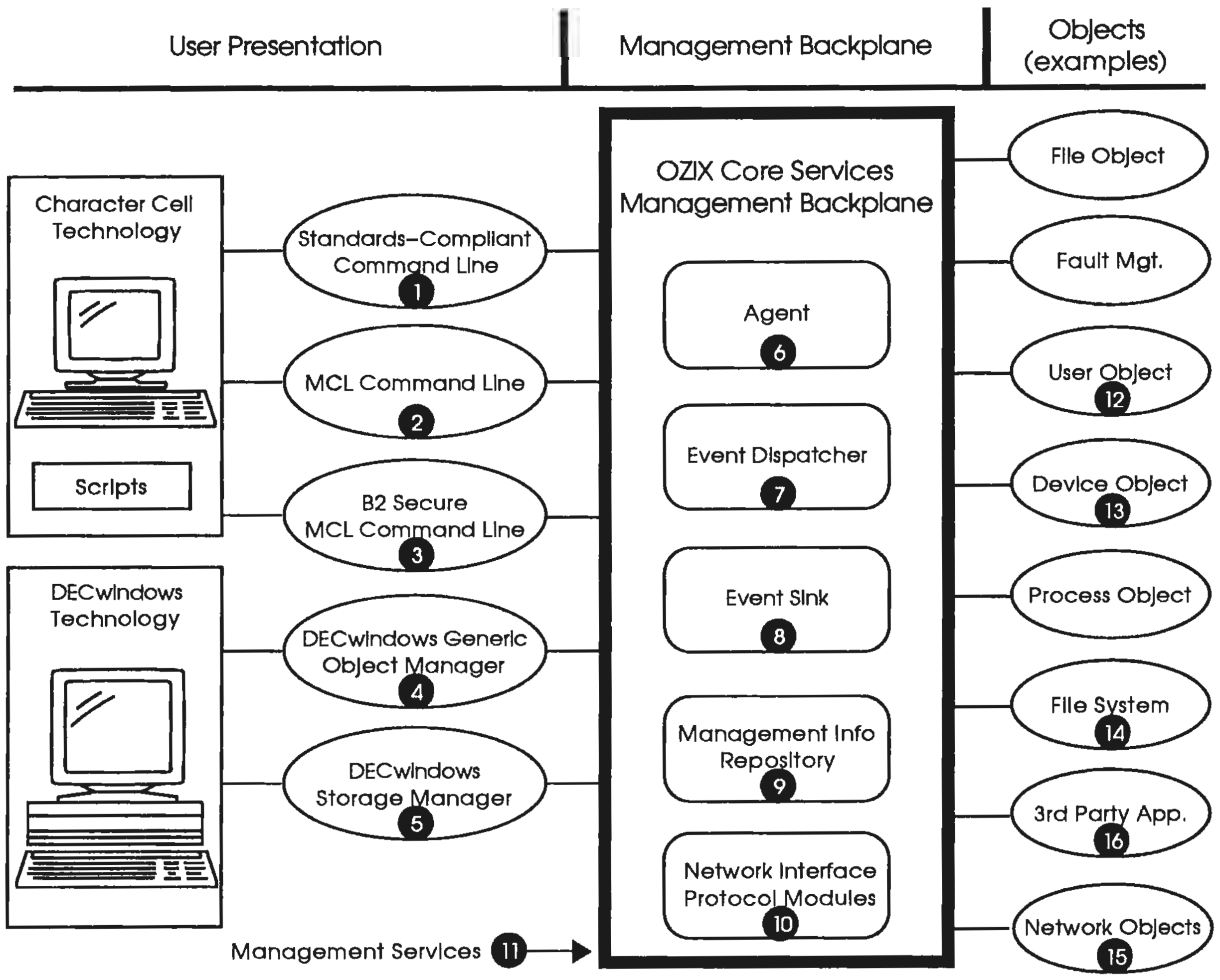
  Developers of these applications must choose how the applications fit into the management framework. As illustrated in Figure 9, third-party applications choose between connecting directly to the Enterprise Management Architecture (EMA) for a Digital specific solution or connecting with the management backplane through network protocols and standards. Each has its advantages and disadvantages.

  EMA provides added-value to the third-party application in that EMA provides better integration with Digital applications and a richer management application development tool set. The disadvantage is that the applications are specific to a Digital system. The converse of the above is true if the third party chooses to connect to the management structure using network protocols and standards.

3.6 OZIX System Administration Component Overview

This section provides the reader with a road map of OZIX system administration components and the specifications. The specifications describe details about each component. Figure 10 illustrates the components of system administration. The objects listed are only examples, unless otherwise stated.

Figure 10: System Administration Components



❶ Standards-Compliant Command Line

Command lines that are being provide to meet industry standards. Documented in the *OZIX System Administration Functional Specification*, *OZIX OSF Commands*, and *OZIX BSD Commands*.

❷ MCL Command Line

Generic command line to manage any manageable object. Documented in the *OZIX System Administration Functional Specification* and *Digital Network Architecture Network Control Language Specification, Version T1.0.0*.

❸ B2 Secure MCL Command Line

Generic command line to manage any manageable object in a B2 secure environment. Documented in the *OZIX System Administration Functional Specification* and *Digital Network Architecture Network Control Language Specification, Version T1.0.0.*

❹ DECwindows Object Interface

Generic object window manager DECwindows Interface. Documented in the *OZIX System Administration Functional Specification.*

❺ DECwindows Storage Manager

Specialized ABA DECwindows manager. Documented in the *OZIX System Administration Functional Specification.*

❻ Management Backplane Agent

Provides object operation dispatching. Documented in the *OZIX Management Agent Functional Specification.*

❼ Management Backplane Event Dispatcher

Provides event dispatching to event sinks. Events are posted to the event dispatcher which then dispatches the event to the appropriate event sink. Documented in the *OZIX Event Dispatcher Functional Specification.*

❽ Management Backplane Event Sink

Provides a "buffer" for event messages. Management applications interested in receiving events subscribed to the event sink. Documented in the *OZIX Event Dispatcher Functional Specification.*

❾ Management Backplane Management Information Repository

Database of object class definitions. Documented in the *OZIX System Administration Functional Specification.*

❿ Management Backplane Network Interface Protocol Modules

Provides for the extension of the management backplane across multiple machines on the network. This extension is know as *distribution services*. This is transparent to the clients of the management backplane. Documented in the *OZIX Management Agent Functional Specification.*

⓫ Management Services

Sum of the services provided by the management backplane. These services hide the fact that there are different components of the management backplane. There are core services and extended services. The core services are documented in the *OZIX Management Services Functional Specification.* Extended services are undefined in this version.

⓬ User Object

Represents the abstract of users and groups. Documented in the *OZIX System Administration Functional Specification.*

⓭ Device Object

Represents devices and containers. Documented in the specifications for the I/O subsystems which implement the objects.

⓮ File System

Represents the file system and data spaces.

⓯ Network Objects

Represents the manageable entities in the network. Documented in various network specifications. Refer to the Network Implementation Architecture chapter.

⑯ Third-Party Applications

Provides for applications that wish to connect to the management backplane. Documented in the *OZIX Guide to Writing Manageable Objects*.

In addition, there are other OSG wide documents describing the overall strategy and architecture. Refer to the *OS/SB Enterprise Management Approach: An Overview*. EMA is documented as part of the DECnet Phase V architecture. Refer to the *Enterprise Management Architecture: General Description #EK-DEMAR-GD*.

CHAPTER 4

RESOURCE/FAULT MANAGEMENT

4.1 Introduction

The major goal of OZIX Resource/Fault Management is to achieve OZIX product's availability, reliability, configuration and serviceability goals. OZIX Resource/Fault Management addresses:

- Error Detection Strategy
- Error Logging
- Fault Analysis
- Control the System Configuration

OZIX Resource/Fault Management concerns both hardware and software.

OZIX Resource/Fault Management is a single system strategy for error analysis and configuration; the strategy has been designed so that a higher level of management could be layered on top of the current single system strategy to manage a group of OZIX systems as a single system. OZIX support for management of multiple systems as a single system is not discussed in this paper, but may be addressed in a future version of OZIX.

4.2 Goals

OZIX product goals from "OZIX Product Features Document" for Resource/Fault Management are:

- Availability - One crash requiring operator intervention per year
- Configuration - Remove, detect, configure and bring-on-line devices
- Reliability - One Hardware/Software induced crash shutdown per year
- Serviceability - Maximize ability to repair and minimize time to repair

4.3 Overview

There are many components of OZIX resource and fault management.

OZIX resource management components are:

- Resource Manager
- User Interface

OZIX fault management components are:

- Error Monitor

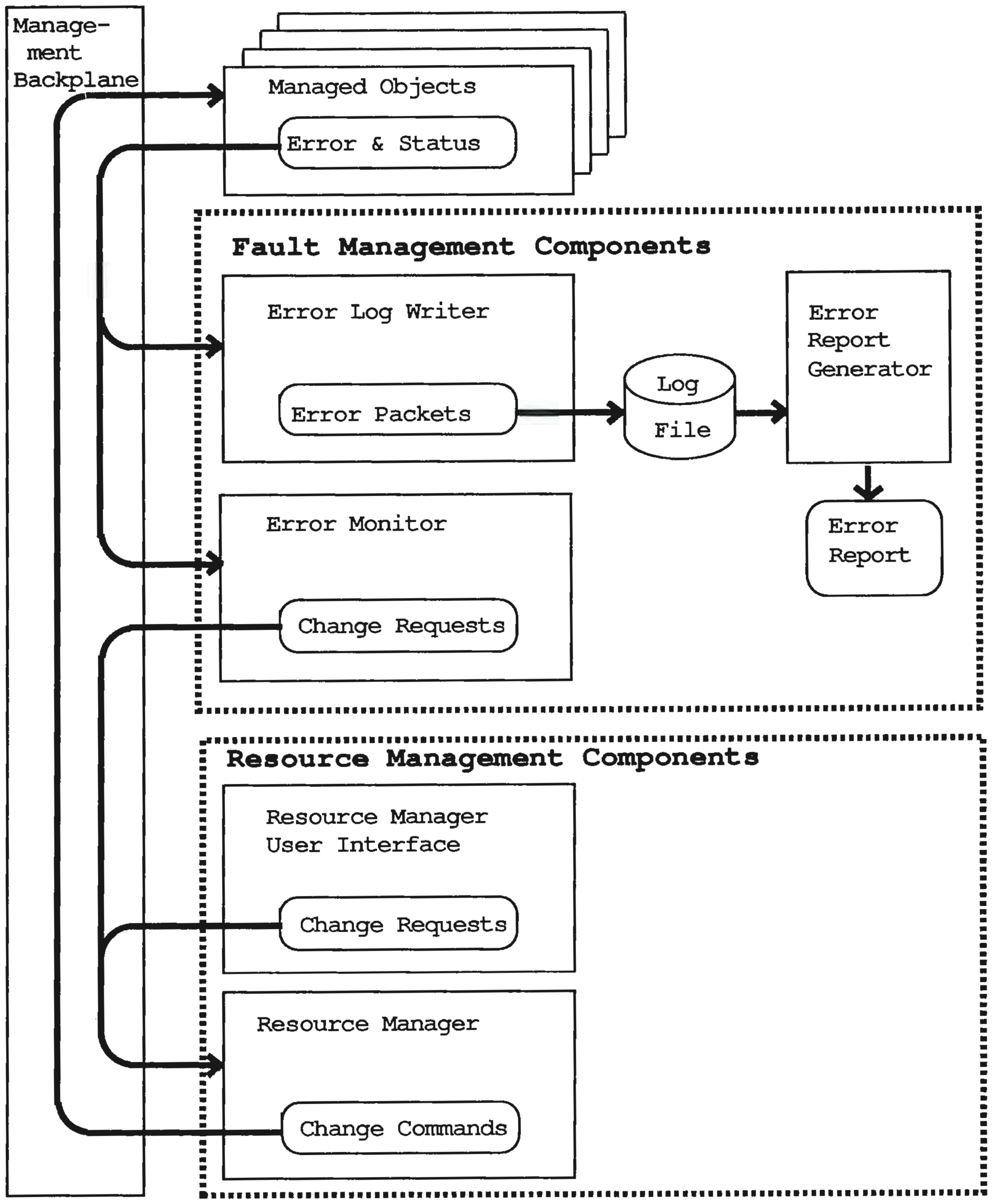
- Error Log Writer
- Error Report Generator
- User Interface
- Crash Dump Writer and Analyzer

Manageable objects work with both the resource and fault management components to achieve OZIX's goals.

The following diagram depicts the components involved in resource and fault management, how information flows between the components. OZIX System Administration Management Backplane is a secure channel over which the resource and fault management components communicate. The System Administration chapter describes the management backplane and its services. A simplified flow of information is:

1. **Error Information** is sent from the **Manageable Objects** to the **Error Log Writer** and the **Error Monitor**
2. The **Error Log Writer** writes the **Error Log File** on demand.
3. The **Error Report Generator** reads the **Error Log File** and writes the **Error Report**
4. **Change Requests** are sent from the **Error Monitor** and **User Interface** to the **Resource Manager**.
5. **Configuration Change Commands** are sent from the **Resource Manager** to the **Manageable Objects**

Figure 11: Resource/Fault Management Overview



4.4 Manageable Objects

Manageable objects are at the beginning and end of the resource/fault management process. Manageable objects detect problems and initiate actions by issuing asynchronous error events to the management backplane. Management operations to correct problems are issued by management applications through the management backplane to manageable objects. An example of a reported error event is "a page of memory is getting soft errors" and an example of a management operation is "remove a disk from the configuration". Note that the manageable object which reports the error event is not guaranteed to be the same manageable object which receives a management operation to correct the problem. Examples of manageable objects are:

- Processors
- Memory
- Buses
- I/O Devices
- OZIX Software Subsystems
- Layered products

Some manageable objects contain error repair logic, fault analysis algorithms and recovery mechanisms. These manageable objects are built this way for various reasons. Some error and fault actions are time-critical and can not tolerate the higher latency of the resource/fault management components. An example of error recovery is retrying a failed operation. Although these manageable objects cover functions "normally" done by other resource/fault management components, the model presented here is still valid. It is, however, important that faults, errors, and actions are reported.

4.5 Error Detection

The starting point for any fault and resource management system is detecting errors. When a fault is stimulated the result is an error. The detection of this error is the act of identifying the difference between what is expected and what is actually observed. In order to meet OZIX product goals, both the hardware and the software must be capable of identifying, handling, and reporting error conditions. The next sections briefly identify some commonly used error detection techniques for hardware and software. These are by no means the only techniques available to the designer.

4.5.1 Hardware Error Detection

Even the best hardware has the property of breaking at a definable failure rate. Hardware engineers design mechanisms into the product to detect a percentage of these hardware errors. OZIX detects these errors by having the subsystems which operate directly with the hardware (the device drivers and kernel nub), look for error information reported by the hardware, and act on this information. Acting on the information includes:

- Saving the error information.
- Recovering from the error if the software can attempt a retry (or other methods).
- Reporting the error information and the ending status of the affected operation to the error logging and monitoring components.
- Reporting the error status to the requester of the operation.

For the error monitor to work it is very important that all errors are reported, including errors which have been handled.

4.5.1.1 Unreported Hardware Errors

The strategy to detect hardware errors not reported by the hardware is:

- Use global techniques such has *timeouts* to detect some additional error conditions.
- Look for any additional detection mechanisms that may help improve the detection for a specific device.
- Select the hardware that OZIX supports based on its hardware failure detection rate.

4.5.1.2 Hardware Design Faults

Besides breaking, hardware also has design faults. Some of these faults are detected by the same detection methods that detect broken hardware. For example, a signal integrity problem on a data bus might cause one bit to be misread which could be detected by the parity or error correction code (ECC) system on the bus. Design faults of this nature will be detected and handled as if they were broken hardware. Some design faults cannot be detected by operating system software and are the province of hardware Design Verification Testing (DVT).

4.5.2 Software Error Detection

The detection methods for software are very similar to those in hardware. For example, parity memory adds an additional bit to each byte of data, defines rules for accessing the data, and the values that indicate an error. In software, error detection is accomplished by having some redundant states and knowing which states are error states.

In general, the detection of software errors is part of the design of each OZIX component. Some examples of the techniques that will be used are:

- Verify that routine arguments are reasonable.
- Have unknown parameters force execution through the "default" case of switch statements and to error handlers.
- Test for queue corruption by checking the type of data structure when it is removed from queue.
- Test for pool corruption by marking memory.
- Validate global variables.
- Monitor critical activities, so that critical resources are not being held indefinitely.

It is very important for the error monitor to work, that all errors are reported, including errors which have been handled.

There are a few "global" software detection mechanisms. In cases where errors are not detected the instant they occur, it is important to contain and isolate their effects. Within OZIX, the subsystem model forces containment of errors to the code that caused the error. This containment is forced in a much higher degree than in past operating systems.

4.6 Error Log Writer

The error log writer records error information. OZIX is using a generalization of the event dispatcher from Phase 5 of DECnet for dispatching error information. Since the needs of error logging are slightly different from network event logging, a few additional features are needed:

- Buffers for certain critical errors types must be locked in physical memory to be easily locatable in the crash dump. When OZIX crashes the error information must be recovered and written to the error log on the next reboot.
- Since the normal case for logging errors will be on the local system, some optimizations are needed to lower local error logging latency. These include treating the Local Sink and the Error Log Writer as special cases and:
    - Pre-configuring them for quick start up.
    - Bypassing DECnet.
    - Bypassing the conversions to and from ASN-1

This error/event dispatching mechanism provides a system that works well in the local case. It also allows the user to tailor a large complex of servers and workstations to use either centralized logging, local logging, or some combination based on device and error code. This also provide a port for any Field Service value-added service to monitor error information.

On each node the local Error Log Writer writes the error and event information to a local error log file. Source information includes the node so that a single file can contain data from many nodes.

4.7 Error Report Generator

The OZIX Error Report Generator is a support tool required for system fault analysis. It allows the system administrator or customer services engineer to obtain detailed information about events on the system and to create a report that is easy to read and interpret.

The Error Report Generator provides the user with a powerful troubleshooting tool by allowing the user to select and report related events. These events may be related by time range, event class (CPU, memory, network, system messages, etc.), device type, and logical or physical device name, as well as several other relations. It is possible for the user to correlate events that were flagged as related when they were written to the log. For example, if a device error causes both an MSCP error and an error at the file subsystem level, and these errors were flagged as related in the log, the Error Report Generator would write both of these entries when either was requested. (With all current Digital report generators, any such correlation must be made manually by the user.) Remote logging is an integral part of the error logging strategy and the log may contain event entries from many different nodes. It is possible to create reports of events on specific nodes.

The output format is also flexible. The user select entries to be written in several forms, from raw hex data to full bit-to-text translation of the entire record. The default output mode selects the most useful information from an entry and reports it in a concise form. A summary report is also available, providing counts of events over the specified range of time.

The Error Report Generator is a user application and reads the log file(s) written by the Error Log Writer. It selects specific entries from this file according to parameters supplied by the user. The selected entries are then formatted and a report written according to user specification.

The main goals of the Error Report Generator include:

- Accurate bit to text translations

- Intuitive and user friendly interface
- Modular design for the easy integration of additional event classes
- Good Performance - the user should not experience excessive delays
- Reliability - recovery and/or report of all errors incurred by the Error Report Generator such as error in input files, lack of resources, or user input.

The Error Report Generator is written in a highly modular fashion. This makes it possible to separate the bit-to-text translations of individual events from the rest of the code, allowing additional devices to be added easily in the future. Today, a major coding effort is required to add devices to the error log. In OZIX it should be possible to add additional devices and events by entering their entry definitions at a higher level and not by modifying and recompiling the source code.

4.8 Error Monitor

The Error Monitor has responsibility for fault analysis in an OZIX system. It is both a management application and a manageable object. As a management application it receives error and state change events from the management backplane and reacts by issuing management operations to the resource manager to correct an error or react to a state change.

As a manageable object it accepts management operations so that prediction algorithms, thresholds, and other decision making parameters can be modified within the error monitor.

The error monitor consists of a data base of errors and rule-based failure prediction algorithms. As errors are received, they are placed in the data base and the rules are tested to see if any conditions have been satisfied. Prediction algorithms in the rules can be simple thresholds like "a memory page is bad if 100 correctable read errors (CRD) memory occur in 1 hour in the page". Once a failure prediction is made, a request is sent to the resource manager to change the configuration to remove the component.

The error data base in the error monitor has knowledge only of manageable object sending errors. The database does not describe the configuration; the database that describes the configuration is maintained by the resource manager.

The rules in the error monitor are not in code but are maintained as a data file. A set of rules are supplied as part of the OZIX system. This allows addition and changes without requiring the error monitor to be recompiled. The error monitor rules handle devices at the physical level and not at logical or virtual levels.

Complex prediction algorithms can be built into the rules. These algorithms can correlate errors from different manageable objects to detect a fault which crosses manageable object boundaries. An example of such a correlation algorithm is determining a bus failure by looking at the errors of all the manageable objects connected to the bus. Multiple rules can be run on the same manageable object. For example, one rule could be thresholding on recovered errors on a given device and another rule could be thresholding on the total of all types of errors.

A rule can take different actions. The simplest action would be to tell the resource manager to remove a component from the configuration. Examples of complex actions would be to initiate diagnostics and exercise programs to attempt to stimulate and determine a fault. Other actions could include writing error information to stable storage on a board (EE PROM) and generating a service alert.

When the error monitor returns error information to the user interface, it does not return simple counts. Instead, the error values maintained by the rules may be displayed.

4.9 Resource Manager

The Resource Manager is responsible for configuration and recovery policy. It is both a management application and a manageable object. As a manageable object it receives management operations, some of which come from user presentations and others from the Error Monitor, to modify the configuration of the system. Additionally, the rules the Resource Manager uses to modify the system configuration are maintained through the manageable object interface the Resource Manager exports to the management backplane.

As a management applications it issues management operations to manageable objects, such as devices, in order to modify the configuration of the system. The decisions made by the Resource Manager as a management application are based on rules maintained by the Resource Manager.

The resource manager consists of a data base which describes the entire configuration, and rule-based recovery algorithms. As requests are received, the rules which apply to this request are tested. The management operations are sent to the manageable objects which are needed to implement the actions of the satisfied rules. The resource manager deals with devices at the physical level and not at the logical or virtual levels.

Resource manager rules are kept in a data file and not in code. A set of rules are supplied as part of the OZIX system. This allows addition and changes to be made to the rules without requiring the resource manager to be recompiled. Layered products such as Rdb™ Star, DECxTP can modify the rules to suit their environmental needs. Rules translate requests into management operations for the manageable objects. Resource manager rules can also make consistency checks, such as making sure the change will keep the system above the minimum configuration and checking for hardware and software version compatibility.

The resource manager database has information about all the elements in both the hardware and software configuration. Entries in the database contain information such as serial numbers and version numbers and other descriptive information. Also contained in the database is a history of recent changes. This allows analysis to be done between the current configuration and a previous configuration.

The resource manager replies to the request allows success or failure of fault recovery action to be returned. When the error monitor generates the service alert, it can then be determine if urgent service is required.

An important duty of the resource manager is version control. This is done by storing the revision information in the database and having rules which insure revision compatibility of components.

4.10 Recovery Mechanisms

Each OZIX component uses different recovery mechanisms. The OZIX system supplies common mechanisms, such as the transaction primitives. OZIX is also designed for recovery. For example, subsystem design encourages isolation and allows some subsystems which fail to be reloaded without the system going down. Each component can also implement its own mechanism and concept, including:

- Removing bad memory pages
- Disk shadowing
- ABA's movement of data from failing devices
- Fault recoverable based file systems

OZIX's design includes quick system restart and recovery.

Again it is very important that the manageable objects report actions taken.

4.11 User Interfaces for Resource/Fault Management

The Resource/Fault Management provides no user interface, per se. Instead, the manageable object interfaces exported by the Resource Manager and Error Monitor are presented to the human through user presentations provided by system administration. These interfaces include both a generic manageable object interface and a specialized user presentation for configuration of an OZIX system. For more information on the user presentations provided, see the System Administration chapter. Through these user presentations, the system administrator will not only be able to modify the configuration of the system, but also the rules used by the the Error Monitor and Resource Manager to configure the system.

In general, manageable objects, once taken off line by the resource manager, will only be placed back on line with intervention by the system administrator. This is done to track system configuration changes and the reasons for returning the manageable object to service.

The OZIX internationalization guidelines are followed in the design of the user interface.

4.12 Crash Dump Writer

When the OZIX operating system detects an internal inconsistency from which it can not recover, such as a corrupted data structure or an unexpected exception, it activates the system crash mechanism. This mechanism writes predetermined state information to dump file(s) and shuts down the operating system in a controlled fashion. This mechanism is also responsible for saving error information, in a previously allocated location. This information is sent to the event dispatcher to be recorded in the error log file upon system reboot.

The OZIX Crash Dump Writer does not rely on the failed operating system to perform any functions on its behalf. It is self contained and thus safe. To ensure that it has not been corrupted, the OZIX Crash Dump Writer code is checksummed prior to writing the crash.

The OZIX Crash Dump mechanism follows a hierarchical or multiple level approach to ensure that enough state information is saved. The system administrator has control over what level of dump is to be performed.

The OZIX Crash Dump mechanism writes the minimum amount of state information required to identify the failure to a small primary crash dump file that is located on the system disk. The primary file is not dilatable and is always present. All other selected levels will be written to a much larger secondary crash dump file. The secondary crash dump file size and location are variable and left up to the system administrator's discretion. The location can be any volume that is known to the OZIX operating system and is set by the system administrator. Management operations will be provided to manage the size and location of the secondary crash dump file.

A path to an alternate crash dump file is provided. This redundant path is used when the Crash Dump Writer is unable to access the primary path. This could be due to the device being off-line or inaccessible to the writer. Failover to the secondary path occurs transparently and does not require operator intervention. The OZIX Crash Dump Writer writes as much information as the file size will allow. When the Writer detects size limits it notifies the system administrator.

To avoid inconsistencies between the primary and the secondary crash dump files the Writer places a tag at the beginning of each file that is later matched by the analyzer. This will ensure that the files are synchronized. The Crash Dump Writer also places system image information, such as version number, into the crash dump file. The Crash Dump Analyzer then uses it for consistency checks with the image information it possesses.

4.13 Crash Dump Analyzer Description

The crash dump analyzer is a special version of the OZIX System Debugger with modifications that allow it to operate on the crash dump files and a running system memory in a read-only fashion. Refer to the detailed description of the OZIX System Debugger for further information.

The OZIX crash dump analyzer has an applicable subset of the system debugger's functionality along with crash analysis specific commands. Included in this is:

- At least ULTRIX MIPS™ dbx functionality
- Compiled language source display, when available
- Symbolic addresses
- Help
- History
- Script and log files
- Formatted data structures
- Language switchable (eg. Machine language translation)
- Format and display different stacks
- Display the locations and contents of procedure call frames
- Validation of the integrity of a queue by checking the pointers in the queue
- Consistency checks
- Display information about the state of the system at the time of the failure. This would include:
  - Date and time of the crash
  - Name and version of the O/S
- Display subsystem specific information
- Display executor specific information

The OZIX Crash Dump Analyzer operates in a client server relationship with the server portion residing in a specialized version of the system debugger nub. For communication the nub uses the OZIX Debug Protocol. Analysis is performed from either a remote node or locally from the rebooted system. This enables the OZIX Crash Dump Analyzer to have full operating support for reading source files, extracting symbols from system images, and displaying windows. For source level debugging all source files and system images must be accessible to the Crash Dump Analyzer. This method of crash dump analysis supports OZIX distributed environment.

CHAPTER 5

SECURITY

5.1 Scope

This chapter addresses the *security* and *integrity* issues involved in the design and development of OZIX. It discusses the security and integrity goals of OZIX, defines the elements of OZIX security and integrity , and describes the design features which allow the implementation of a secure OZIX operating system.

References are made throughout this chapter concerning *B2 Certification.* It is not expected that V 1.0 of OZIX will be either certified or certifiable. A later release of OZIX will be certified. However, the entire development and design structure to support certification will be in V 1.0. The elements missing for certification will be primarily documentation, security testing, and some support tools.

Throughout this chapter, new or unusual terms will be italicized on their first occurrence, and their definitions will be found in the glossary.

5.2 OZIX Security Goals

There are three broad sets of goals of the OZIX security design: basic security and integrity goals, extended security and integrity goals, and the global goals.

The basic security and integrity goals of OZIX include:

- Produce a system with significantly enhanced security, integrity and *robustness* relative to existing commercial systems.
- Be certifiable to the B2 level by the *National Computer Security Center (NCSC)*, as defined by Department of Defense (DoD) 5200.28.STD *(the Orange Book)*.

The extended security and integrity goals of OZIX include:

- Support multiple different security and integrity models that are selectable by the customer. This will allow the *access controls* enforced by OZIX to model, as closely as possible, the customer's policies of accessing data.
- Allow the customer, at his discretion, to reduce the level of *enforcement* to enhance usability and performance.
- Provide the features required to permit applications running on OZIX to define and enforce their own access controls and to be incrementally certified.
- Be compatible with various models of distributed security.

The global security and integrity goals of OZIX include:

- Produce a high performance secure system

- Produce a highly usable secure system
- Produce a secure system whose security and integrity features are easily manageable

5.3 Basic Security and Integrity Goals

The security and integrity goals of a commercial operating system are:

- Preserve the *confidentiality* of sensitive information. Confidentiality is obtained by preventing sensitive information from reaching users who do not have a *clearance level* that permits them to handle such information. This type of protection is known as *Mandatory Access Control (MAC)* because it cannot be overridden by the owner of the information.
- Preserve the integrity of valuable information. Integrity is obtained by preventing valuable information from being altered or destroyed except by users at a *reliability level* that permits them to do so. This type of protection is known as *Integrity Access Control (IAC)* because it is designed to ensure the integrity of information.
- Allow the owner of data to restrict the distribution of his data, even from those with adequate clearance. This type of protection is known as *Discretionary Access Control (DAC)* because this control is applied at the discretion of the owner.
- Prevent any user or users from denying access to the system or its resources by authorized users. This type of protection is known as *Assurance of Resources (AOR)*.

Not only must the system satisfy these requirements in normal usage, but it must do so even when one or more *malicious* users are trying their worst to bypass the system protections.

MAC, DAC, and IAC can be thought of as three layers of access filtering that occur on any access attempt. The algorithms used by these layers to make their access decisions are the *security model*, *discretionary model*, and *integrity model*, respectively.

OZIX will be designed and developed to provide these features. In order to have an independent assessment of the security and integrity strengths of OZIX, and to provide our customers with assurances of the security and integrity features of OZIX, OZIX will be certified at the B2 level.

The B2 level of certification is considered appropriate because it provides a level of security and resistance to penetration that is adequate for all but sites involved in areas of national security.

The B2 level of certification is not considered excessive in terms of costs or other impacts, because it is not extremely difficult to validate and it imposes no limitations on export sales.

B2 Certification will assure that:

- A controlled development process and good software engineering techniques will be used. These factors are known to help create a more error-free system.
- A well known set of security related features as defined by the Orange Book will be implemented. These features will provide what is considered a minimum set which can assure system security.
- The strength of the system's security is quantifiable. A B2 system has a well defined strength, and can be compared to the strengths and assurances of other systems.
- The system's strength and resistance to penetration has been tested and proven by a board of expert independent evaluators.

These assurances are independent of the particular uses made of them. They can be used to implement a pure B2 system using the DoD model, or they can be used to implement a commercial system with the strength of a B2 system.

Most commercial customers would find a pure B2 system to be inappropriate for their needs. For this reason OZIX will implement an extended security and integrity model to meet the needs of our commercial customers, reserving the pure B2 model for those few applications where it is needed.

5.3.1 Implementing the Basic Protections

To state the requirements of a secure operating system more precisely: a secure operating system must control the access of *subjects* (things that perform actions) to *objects* (things upon which actions are performed) and *resources* (things used to perform actions).

This control is accomplished by:

- **Identifying** the fundamental subjects, objects, and resources
- **Isolating** all subjects from automatic access to any objects
- **Granting access** explicitly to appropriate objects
- **Controlling** usage of shared resources
- **Trusting** all code that implements isolation, access granting, or that deals with shared objects or resources
- **Recording** all accesses for later review to identify potential attacks on the system

The identification of trusted code leads to an additional access control constraint: data that is critical to maintaining the basic security and integrity of the system must be accessible only to code that is trusted enough to have that access. This access control is known *Trusted Access Control (TAC)*.

5.3.1.1 Identifying the Fundamental Entities

The fundamental entities that are relevant in OZIX are: subjects, objects and resources.

- Subjects include the following:
  - Executors—An *executor* is the OZIX generalization of a process encompassing all *threads* acting together in a single *execution context*. An executor has an identity and a clearance level that cannot be changed during its existence.
- Objects include the following:
  - Dataspaces—A *dataspace* is the OZIX conception of long-term persistent data on the system and is potentially accessible to any executor. Dataspaces are one of the primary security objects to which access must be controlled.
  - Sharable Memory Segments—Memory segments that can be mapped to the execution contexts of multiple executors are potential channels for data flow, and must have their access controlled.
  - Devices—Physical devices such as tapes, disks, printers and networks can act in some sense as a data repository. Access to these devices must, therefore, be controlled.
- Resources include the following:
  - CPU capacity–the compute power of the system
  - Memory space–physical memory pages
  - Disk space–file system, cache and paging space
  - I/O bandwidth–primarily the rate at which data can be moved to and from disk

- Network bandwidth–the rate at which data can be moved over the networks
- Physical devices–disks, tapes, terminals and network connections
- All other limited resources–anything shared and of fixed size or number

5.3.1.1.1 Labeling of Subjects and Objects

To implement access control, all subjects and objects must be labeled. These labels contain the identity of the subject or object, and all information required to make access decisions. These labels are located in memory areas not accessible to untrusted executors.

The elements that make up a label are:

- Unique Security Identity
- *Sensitivity Level* Range and *Compartments*
- *Integrity Level* Range and Compartments
- Ring Brackets

5.3.1.1.1.1 Unique Security Identity

Each security subject and object is assigned a unique identity. This identity is unique over the life of the system, and unambiguously identifies the subjects and objects in the audit log.

5.3.1.1.1.2 Sensitivity Levels

The sensitivity level of an object is usually used to determine what subjects can access an object, and in what manner. For example, a user cleared to the secret level would not be allowed to read top-secret data.

The sensitivity level is a hierarchical range, varying from highly sensitive (for example, top secret) to not sensitive (for example, unclassified).

OZIX will support 256 sensitivity levels. The human-readable labels associated with each level will be able to be set by the customer.

5.3.1.1.1.3 Sensitivity Compartments

Sensitivity compartments are usually used to control access to data on a "need-to-know" basis. For example, a user cleared to access security data would not necessarily be allowed to access payroll data.

Sensitivity compartments are a non-hierarchical set of classes of information. An object may belong to one or more of these classes. OZIX will support 1024 sensitivity compartments. The human-readable label associated with each compartment will be able to be set by the customer.

5.3.1.1.1.4 Integrity Levels

The Integrity level of an object is usually used to determine what subjects can access an object, and in what manner. For example, a user reliable at the malicious level would not be allowed to modify system-critical data.

The integrity level is a hierarchical range varying from highly reliable to malicious for users and from precious to junk for data.

OZIX will support 256 levels of integrity. The human-readable label associated with each integrity level will be able to be set by the customer.

5.3.1.1.1.5 Integrity Compartments

Integrity compartments are usually used to control access to data on a "need-to-modify" basis. For example, a user cleared to modify security data would not necessarily be allowed to modify payroll data.

Integrity compartments are a non-hierarchical set of classes of information. OZIX will support 1024 sensitivity compartments. The human-readable label associated with each compartment will be able to be set by the customer.

5.3.1.1.1.6 Security Rings

Security rings are used to limit access to system critical data to code that is trusted to access the data.

Security rings are a set of heirarchical levels ranging from 0 (completely trusted) to 255 (completely untrusted). All security objects in the system are labeled with two ring numbers: a read ring and a write ring. Any executor whose current ring execution level is at or below the data's read ring level is permitted to read (or execute) the data. Any executor whose current ring execution level is at or below the data's write ring level is permitted to write the data.

An executor's current ring execution level is determined by the *ring brackets* (read ring level and write ring level) of the code the executor is executing. Executing code may set its executor's current ring execution level to any value equal to or greater than that code's write ring level.

For example, if an executor is executing at a current ring execution level of, say, 50, then that executor could only execute a *Subsystem Procedure Call (SPC)* to code whose read ring level was 50 or higher. If a call were made to code whose ring brackets were [50,20], that code could set the executors ring execution level as low as 20. Upon return from the called code, the executor's ring execution level would be returned to 50.

Figure 12 illustrates how ring levels apply to OZIX *subsystems*, *packages* and data. For the sake of simplicity, this example will ignore the other access controls on the packages (MAC, DAC and IAC).

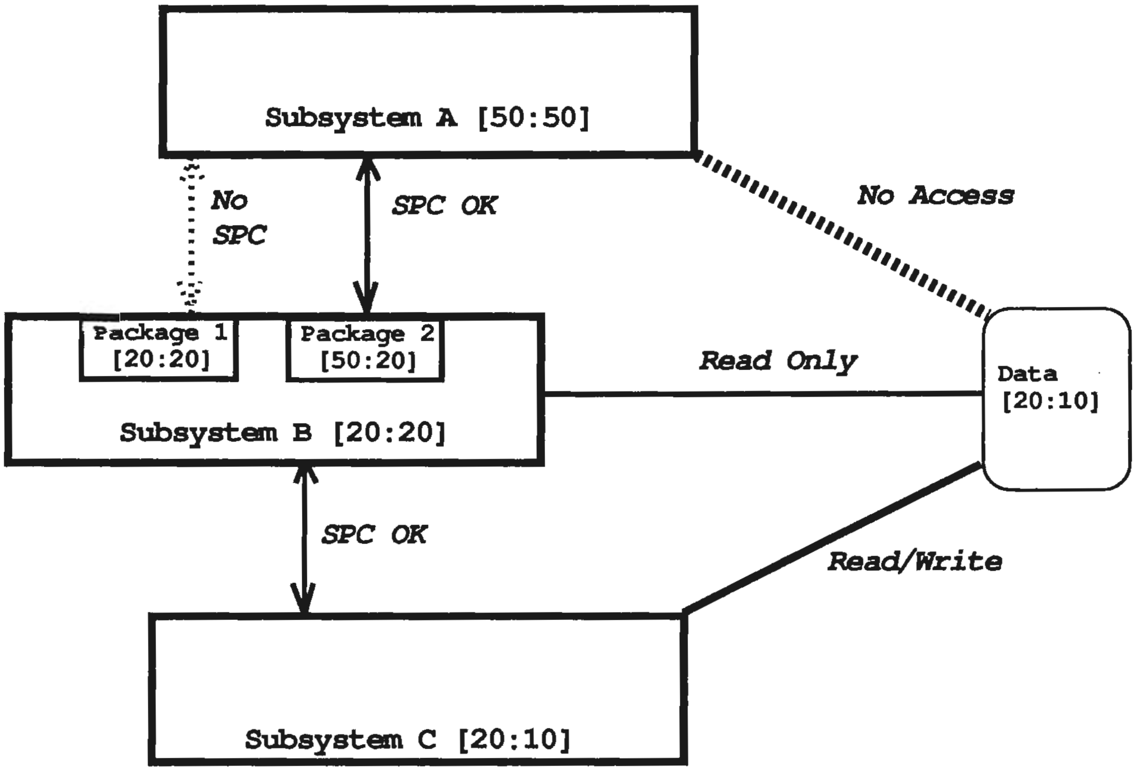
Three subsystems are illustrated: A, B, and C. Subsystem A has ring brackets of [50:50]. Subsystem B implements two packages: package 1, whose ring brackets are [20:20]; and package 2, whose ring brackets are [50:20]. The body of the subsystem B code is at ring brackets [20:20]. Subsystem C has ring brackets of [20:10]. Also shown in Figure 12 is a data object with ring brackets [20:10].

An executor in subsystem A is running relatively untrusted code at ring level 50. No executor in this subsystem can access the data object. An executor in subsystem A may not call subsystem C directly at all. Package 1 of subsystem B also cannot be called, though package 2 can be called.

Once an executor has entered subsystem B, the code in subsystem B can lower the ring execution level to as low as 20. At ring level 20, the data object can be read, but cannot be written.

Once at ring execution level 20, a call can be made to subsystem C. Subsystem C can lower the ring execution level to 10. At level 10 the data object can be both written and read.

Figure 12: Security Ring Levels



5.3.1.2 Ensuring Isolation

The actual isolation in OZIX is provided by the hardware protection of memory. All other isolation mechanisms are built on top of this memory protection. The software isolation mechanisms are designed such that any executor attempting to violate the security or integrity models will violate the hardware memory protections and be vectored to the *Nub*.

Executors are isolated from each other's data because each executor executes in a separate execution context. (Loosely, an execution context can be thought of as an address space). It is impossible for an executor to reference any memory that is not in its execution context. Any address that an executor can state is within his own virtual address range. If no physical memory is at the specified address, or if the access mode is not appropriate for the specified address, the hardware protection mechanisms will trap the access attempt.

This isolation by execution context extends to much more than the virtual addresses that can be seen by application level code. Many of the subsystems that implement OZIX use virtual memory segments that are selected depending on the particular executor running in the subsystem. Thus, even these subsystems cannot reference addresses that do not belong to the running executor.

5.3.1.3 Granting Access to Objects

The *access validation component* is the mechanism whereby an executor is granted access to additional objects. A request, for example, to add a shared segment to an executor's execution context will result in a call to the access validation component. The access validation component will apply each of the defined access filters (security model, discretionary model, integrity model, trustedness model) to the request, and, if the access passes all the filters, permit the mapping to occur.

5.3.1.4 Controlling Usage of Resources

The general goal of controlled resource usage is to prevent a user or group of users from denying service to other users. This statement of the goal, however, seems too simple for a system designed to run in a wide variety of environments. In some circumstances it may be acceptable (indeed, required) that a highly critical task be performed, even if it consumes the entire system. In the extreme cases, it may even be acceptable if other, less critical tasks are aborted.

Thus OZIX implements a new executor attribute, *priority*, and restates the goal as: "No user is permitted to deny a user of higher priority access to system resources".

In order to implement this goal, all usage of all limited system resources will be controlled. Some such resources are obvious: CPU cycles, memory, disk space. Others are not so obvious: bandwidths, fixed size system structures, locks.

Any subsystem that controls a shared resource will prevent denial of resources attacks. This implies that usage records, quotas and a priority algorithm will be enforced for any such shared resources.

5.3.1.5 Trusting Code

Code must be trusted either because it must be correct or because it must be discreet:

- Correct code implements the pervasive security features that control isolation and access—such as the access checking mechanisms —such code must be correct or the pervasive security features would not work
- Discreet code, by the nature of its function, must access data from multiple levels or users—such as the dataspace cache that caches data for all users and levels—such code must be discreet since it **could** disclose information.

Such portions of the system must be trusted to perform their functions correctly and not disclose data inappropriately. Trust is not earned easily. In a certified secure operating system trusted code must be developed using good design practices, and its design and code must be carefully reviewed. Trusted code must be subjected to extensive testing and documentation.

Any portion of the system that is trusted becomes a member of the *trusted computing base (TCB)*. Since considerable effort is involved in trusting a piece of code, a goal of OZIX is to minimize the amount of trusted code and, hence, the size of the TCB.

5.3.1.6 Recording Access Events

Various types of security relevant events must be able to be audited by the TCB. The intended function of the audit facility is to permit the security officer to detect and analyze break-in attempts.

The types of events that will be audited include:

- Use of *authentication* mechanisms
- Addition or deletion of objects from a user's address space

- Actions taken by the operator or administrators
- Production of printed output
- Override of human readable output markings
- Change of designation of any communication channel or I/O device
- Events that may exercise *covert storage channels*

Tools to analyze the recorded audit information will be developed.

5.3.2 Obtaining Certification

Obtaining a certification from the NCSC requires:

- Developing the system in accordance with the NCSC's guidelines *(the rainbow books)*
- Designing the system to include the NCSC required security features
- Producing security related documentation
- Verifying that the system meets the NCSC guidelines
- Performing security related testing
- Submitting the system to the NCSC for approval
- Passing NCSC penetration testing

5.3.2.1 Development

At the higher levels of certification, security and integrity must be designed into a system from the start. Add-on security may be acceptable for lower levels of assurance, but this is not true at the B2 level and above. Indeed, the very structure of a B2 system is considered part of the security assurances.

5.3.2.1.1 Methodology

Development of a certified system must follow "good" software engineering practices. This has been generally interpreted to mean using the specification, design, code model with appropriate reviews, inspections and testing.

These software engineering technologies are accepted as providing a system which with fewer errors, more consistent implementation, and better maintainability than systems developed using ad hoc methods.

The development methodologies used on OZIX are documented in the *OZIX Software Development Procedures* listed in [5] in Section 5.7.

5.3.2.1.2 Configuration Management

The documents and code related to a certified system are expected to be under configuration control during the entire life cycle. This is intended to allow traceability of changes (both design and code), and to prevent trojan horses, trap doors, etc. from being inserted into the system.

5.3.2.2 Required Security Features

The Orange Book requires:

- Identification, analysis, control, test and audit of covert channels
- Formal and proven security policy model
- Good development practices including configuration management
- Detailed top level specification (DTLS) for the TCB
- Support for *multi-level devices*
- Mandatory access control for all subjects and objects
- Separate domain of execution for the TCB
- Modularized TCB
- Principle of *least privilege*
- Identification and documentation of all internal and external TCB interfaces
- Secure TCB generation
- Trusted path for login and some other security functions
- Auditing of security relevant events

5.3.2.3 Documentation

The following documents must be produced for evaluation by the NCSC:

- Security Features User Guide–A user document that describes the protection mechanisms provided, guidelines on their use, and how they interact with one another.
- Trusted Facility Manual–A system administrator's manual that describes the administrator and operator actions related to security, guidelines on consistent and effective use of the protection features of the system, and procedures for secure generation of a new TCB from source after modification.
- Test Documentation–A document that describes the test plan, test procedures and results of the testing of the security mechanisms. It must include results of testing the effectiveness of the methods used to reduce covert channel bandwidths.
- Design Documentation.

  The following are required:

  - The TCB interface description.
  - A formal security policy model description.
  - A descriptive top-level specification–A complete description of the TCB including exceptions, error messages, and effects.
  - Miscellaneous documentation.
    - Description of the reference monitor implementation and an explanation of why it is tamper-resistant, cannot be bypassed, and is correctly implemented.
    - Description of how the TCB is structured to facilitate testing.
    - Description of how the TCB is structured to enforce least privilege.

- Results of covert channel analysis and the tradeoffs involved in restricting the channels.

5.3.2.4 Verification

The following techniques must be used to verify the specified characteristics of the system:

- Analyze–the design documentation, source code, and object code.
- Prove–that the formal model of the *security policy* is consistent with its axioms.
- Show–that the descriptive top-level specification of the TCB completely and accurately describes the TCB and provides an accurate description of the TCB interface.

5.3.2.5 Testing

The following security-specific testing is required:

- The security mechanisms must be tested and found to work as claimed in the system documentation.
- Testing must demonstrate that the TCB implementation is consistent with the descriptive top-level specification.
- The effectiveness of the methods used to reduce covert channel bandwidths must be tested and prove to function correctly.

5.3.2.6 NCSC Approval

The overall system development, design, testing and documentation must be approved by the NCSC in order to obtain certification. The NCSC *Vendor Assistance Program (VAP)* provides a review team of evaluators during early system design phases to help a vendor produce a certifiable system by answering questions and reviewing designs and documents.

5.3.2.6.1 NCSC Penetration Testing

Armed with all system documentation (including code), the NCSC will assign a team to attempt to bypass the security features of the system. The penetration team will usually consist of about three experienced evaluators.

Digital will be required to supply a test system for the use of the evaluators, either at our site or theirs.

5.4 Extended Security and Integrity

Meeting the basic security and integrity goals is not enough. A system that met only these goals would indeed be a B2 secure system, but it would not satisfy the real requirements of our customers and application developers. OZIX supplies a set of features that permit a customer or application developer to tailor and extend the security and integrity characteristics of the system to fit the customer's needs.

5.4.1 Alternate Models

For purposes of certification, OZIX will be equipped with a slightly modified version of the Bell and LaPadula Security Model. This model satisfies the requirements of DoD security and its mathematical properties are well known within the security community. This model is not a desirable model for the majority of commercial customers because it excessively limits the capability to share data.

OZIX will permit a customer to install alternative access control models to replace the standard models. These models can be tailored to fit the way that the customer views confidentiality and integrity of data.

Section 5.6.2 describes an alternate set of access controls that might be used by a commercial customer. This simplified model is patterned on the way that Digital handles confidential data.

5.4.2 Alternate Authenticators

OZIX consolidates all authentication in one place, the Access Validation Subsystem, both for added security, and for ease of system management. Since this scheme prevents a customer from simply replacing or adding programs to perform custom authentication, OZIX provides the capability to add additional authentication mechanisms to the Access Validation Subsystem to meet his sites requirements. These mechanisms can be used at login, or for identity authentication from the application level.

For example, if a customer wishes to use *smart card* authentication, a new authenticator which supported this type of identity validation could be added to the access control subsystem.

5.4.3 Enforcement Levels

There are a number of security checks and controls that taken together enforce system security and integrity. OZIX provides the capability to increase performance (at the cost of reduced security and integrity) by selectively disabling these controls. The following are the security and integrity enforcement switches that the customer can disable at his discretion.

- Covert Channel Delays–A value that controls the amount of delay used to limit covert channel bandwidth
- TCB Stacks–Enable creation of a new stack when entering the TCB
- Auditing Control–Audit selections:
  - Levels–The level of significance an event must have to be audited
  - Events–Particular event types that are to be audited
  - Requested–Determines whether to honor ACL requested audit events
- Isolate Address Spaces–Enables isolation of address spaces in separate subsystems
- Device Sanitization–Enables complete reinitialization of devices and terminals when changing labels
- Integrity Access Control–Enables checking of IAC labels upon access
- Exact Quotas–Use *mutexes* when updating/checking quotas
- Quota Checking–Quotas limits for individual resources may be separately selected or disabled
- ID Stack–Enables id stack updates during gate crossings to allow context sensitive IAC
- Discretionary Access–Enables DAC checking using access control lists

- Mandatory Access–Enables MAC label checking upon access
- Trustness Access–Enables TAC label checking upon access

5.4.4 Features to Support Secure Applications

A goal of OZIX is to supply the features required to allow secure applications to be implemented on top of the operating system. Ideally, these applications could be incrementally certified to any level at or below B2.

OZIX implements four features to aid in this goal: *security classes*, *access stacks*, *ring levels*, and *identity stacks*.

5.4.4.1 Security Classes

OZIX permits the creation of security classes. A security class is a set of objects whose accesses are controlled by a set of MAC, DAC, IAC, TAC and other models. It is possible in OZIX to define a new security class whose accesses are controlled by a discrete set of access filters.

Subjects belong to a set of classes and may access objects only in the classes to which they belong.

The capability of defining new security classes permits applications to create new and different types of access controls which will be enforced with the full strength of OZIX's B2 security.

For example, a database application might choose to implement a new class of objects called "database objects". These objects could co-exist with normal objects, while remaining inaccessible to non-database users. The access controls enforced on the database objects might be entirely different from the access controls enforced on the normal objects.

5.4.4.2 Access Stacks

As mentioned previously (Section 5.3), the access controls applied to a security class can be considered as a stack of access filters. OZIX permits the addition of new filters into the access filter stack of any security class.

This capability permits the creation of new and different access controls which are enforced with the full strength of OZIX's B2 security.

Some examples of uses of this feature would include:

- License Access Control—accesses can be permitted or denied depending on whether the access is permitted based on the installed licenses.
- Timed Access Control—access to certain objects may only be permitted at particular times of day, or on certain days
- Extended Auditing—a null access filter (one which always permits access) can be installed which performs additional or special auditing, such as across the network to a write only auditor

5.4.4.3 Ring Brackets

The ring bracket mechanism can be used by applications to implement their own *security kernel*. Just as the ring brackets allow OZIX to protect the files and structures which are required for its correct operation, applications can use ring brackets to protect their own files and structures from everyone except OZIX—and OZIX can be trusted not to corrupt them.

This capability allows, for example, a database application to protect its own user password file with B2 level assurance that users cannot corrupt it.

5.4.4.4 Identity Stack

OZIX supports the concept of an identity stack. This stack can be considered to be a list whose first element is the user's name, and each subsequent element is a component identity. At any instant, the identity stack shows the entire call history of the current execution. For example, if user JIM, running a Korn shell, calls a database application which in turn calls the API subsystem which calls the file name subsystem, during execution in the file name subsystem, the identity stack would be:

JIM.KSH.RDB_STAR.OSF_API.FILE_NAME_SS

DAC checks for object access can perform pattern matching operations on the identity stack. For example, an ACL might permit access to a file only if the user identity were "JIM", and the identity "RDB_STAR" were immediately preceding OSF_API on the identity stack.

This capability allows applications to limit access to objects to only the code which is expected to manipulate them.

5.4.5 Distributed Security

OZIX will not initially support network distributed security. However, the mechanisms for adding this support are available. Alternate authenticators can be easily added to the access validation subsystem and used by applications and system services for distributed authentication.

5.5 Global Goals

Security and integrity enforcement cannot be free of performance impact. Additional computation is required in performing access checks, access to new databases must be performed, and inefficiencies may be introduced by the structuring of the system The challenge is to minimize the visible impacts of security and integrity enforcement.

The design philosophy of OZIX is to "minimize the mainline". That is, not to add security relevant code to the normal execution paths. OZIX will rely heavily on cached access checking. Once a user has been given access to something, that access need not be checked again. OZIX will be designed to contain the majority of the access checking impacts to the initial access attempt, and to have a minimal impact for the duration of processing.

Security and integrity enforcement cannot be free of usability impact. Additional constraints and limitations are imposed, and additional features are supplied which must be managed. The challenge is to minimize the visible impacts of security and integrity enforcement.

The design philosophy of OZIX is to "minimize the additional actions required". That is, not to add security relevant actions to normal execution paths. OZIX will rely heavily on defaults to provide security and integrity information. Once a user has set up his security and integrity configuration, little additional will be required of him.

Data sharing cannot be as unconstrained in a secure system as in an insecure system. The very nature of security and integrity enforcement is to add "cannots" to a system. This is not something that can be designed around. OZIX will minimize the impact of reduced data sharing by allowing the customer to specify what the sharing model is to be, and not force an overly restrictive model on the users.

Security and integrity enforcement can not be free of system management impact. Additional features, attributes and controls are added as a result of security and integrity enforcement, and these must be managed. The challenge is to minimize the visible impacts of security and integrity enforcement.

OZIX will minimize the system management impacts of security and integrity in several ways. OZIX will use a consistent management approach across all aspects of the system, including enforcement. Thus a system administrator will not have to learn to use tools which are unique to managing the system enforcement. OZIX will integrate many of the security and integrity controls into common objects with common interfaces in place of the ad hoc databases used in many systems. This will present a unified interface to the system administrator at a single point, instead of multiple types of interfaces applied to many different features. Finally, OZIX will depend on common standard default enforcement configurations and models, which will permit a customer to install a consistent system with very little tuning or modification.

5.6 Examples of Security and Integrity Models

The following sections will compare and contrast two different security and integrity models with which OZIX could be equipped. The first is a certifiable Department of Defense (DoD) model, the second a commercial security model.

5.6.1 Department of Defense Security Model

This section describes a security model usable by the Department of Defense (DoD) and which will be the basis of the certified OZIX product.

Table 2 defines the security and integrity labels which are used by this access control model.

Table 1: DoD Model Labels

Levels	Compartments
Security Levels	**Security Compartments**
UNCLASSIFIED	NATO
CONFIDENTIAL	CRYPTO
SECRET	NUCLEAR
TOP_SECRET	etc. etc.
Integrity Levels	**Integrity Compartments**
none	none

OZIX will be certified using a modified Bell and LaPadula model for mandatory access control using these labels. This model states:

- **A user cannot read data that is at a higher sensitivity level than the user's own sensitivity level.**

  For example, a user classified at the secret level cannot read top secret data. This restriction is known as the *simple security policy*. It prevents a user from reading data he is not cleared to see.

- **A user cannot write data that is not at the user's sensitivity level.**

  For example, a user classified at the secret level cannot write data into an unclassified file. This restriction is known as the **-property*. It prevents a user from accidentally or intentionally placing data where it could be read by someone not cleared to see it.

- **A user cannot read data unless he is cleared to access all of the compartments of the data.**

  For example, a user classified at the secret level and belonging to the compartments NATO and CRYPTO cannot read secret data belonging to the NATO and NUCLEAR compartments. This restriction is known as *need to know*. It prevents a user from seeing data that is not required to perform his assignment.

- **A user cannot write data that is not in all of the user's compartments.**

  For example, a user belonging only to the compartments NATO and CRYPTO cannot write a file belonging only to the NATO compartment. This restriction prevents a user from accidentally or intentionally placing data where it could be read by someone without the "need to know" that data.

The Bell and LaPadula model contains no rules concerning integrity. In general, the models used for NCSC certification use no integrity models.

5.6.2 Commercial Access Control Example

This section describes a simple access control model which would be applicable in many commercial development environments, that can be implemented using the extended security and integrity features of OZIX.

Table 2 defines the security and integrity labels that will be used by this access control model.

Table 2: Commercial Model Labels

Levels	Compartments
Security Levels	Security Compartments
UNCLASSIFIED	GENERAL
INTERNAL_USE_ONLY	SYSTEM
CONFIDENTIAL_and_PROPRIETARY	FINANCE
	PREPARED_by_ATTORNEY
Integrity Levels	Integrity Compartments
UNKNOWN	GENERAL
LOCAL	PRODUCTION
VITAL	TEST
	DEVELOPMENT
	SYSTEM

The security levels represent the levels of sensitivity common in a commercial corporation.

The Security compartments represent classes of data that may be accessible on a "need-to-know" basis.

The integrity levels represent levels of trust of code, and value of data.

- VITAL code is code that is highly trusted. This usually consists of stable, commercially produced code that is installed by the system administrator. VITAL data is data of high value, whose loss would have significant impact, such as system configuration databases or major work products.

- LOCAL code is code that is somewhat trusted. It is usually produced by normal system users and that is believed to operate correctly. LOCAL data is data of intermediate value, whose loss would have some impact, such as memos or executables.
- UNKNOWN code is code that is potentially malicious or incorrect. This category of code would include code obtained from random outside sources, and known "buggy" code. UNKNOWN data is data of little or no value whose loss would have little impact, such as temporary files.

The Integrity compartments represent classes of data that may be accessible on a "need-to-modify" basis.

Access between subjects and objects are based on these levels and compartments. The rules that define allowable accesses are the security and integrity models.

The security model rules are:

- **A user cannot read data that is at a higher security level than the user's own.**

  For example, a user cleared at the INTERNAL_USE_ONLY level cannot read CONFIDENTIAL_and_PROPRIETARY data. This restriction prevents a user from reading data he is not authorized to see.
- **A user cannot read data unless the user is authorized to access all of the compartments of the data.**

  For example, a user belonging to the compartments FINANCE and PREPARED_by_ATTORNEY cannot read data belonging to the FINANCE and SYSTEM compartments. This restriction prevents a user from seeing data that is not required for his assignment.
- **A user cannot write data that is not in all of the user's compartments.**

  For example, a user belonging to the compartments FINANCE and GENERAL cannot write a file belonging only to the GENERAL compartment. This restriction prevents a user from accidentally or intentionally placing data where it could be read by someone without the "need to know" that data.
- **Low integrity code may not read high sensitivity data.**

  For example, a program of UNKNOWN integrity (imported from a billboard, perhaps) may not read CONFIDENTIAL_and_PROPRIATARY data. This restriction prevents *Trojan Horse* programs from reading and re-distributing sensitive data.

The following are the integrity model rules:

- **A user cannot read data that is at a lower value level than the user's own reliability level.**

  For example, a user at the LOCAL reliability level cannot execute (read) an UNKNOWN reliability program. This restriction prevents a program from an unknown source from acting as a Trojan Horse and destroying valuable data.
- **A user cannot write data that is at a higher integrity level than the user's own reliability level.**

  For example, a user classified at the LOCAL integrity level cannot write data into a file classified at VITAL integrity. This restriction prevents a user of limited integrity from contaminating data that must be highly trusted to be correct.
- **A user cannot write data unless the user is authorized to access all of the compartments of the data.**

For example, a user belonging to the compartment TEST cannot write data belonging to the PRODUCTION and TEST compartments. This restriction prevents a user, who is performing testing, from modifying production data with possibly incorrect code.

Table 3 compares the modified Bell and LaPadula access control model described in Section 5.6.1 and the simple commercial access control model defined here.

Table 3: Comparison of DoD and simple Commercial Access Control Models

DoD Model	Simple Commercial Model
Security Model (Modified Bell & LaPadula)	**Security Model**
Subject may not read higher level object	Subject may not read higher level object
Subject may not write any other level object	
Subject may not read objects in other compartments	Subject may not read objects in other compartments
Subject may not write object with fewer compartments	
Only security officer may downgrade	Object owner may downgrade
	Low integrity code may not read high security data
Integrity Model	**Integrity Model (Simple Biba)**
None	Subject may not read lower level object
	Subject may not write higher level object
	Subject may not write objects in other compartments

The view of security in a commercial environment is quite different that that in a DoD environment. In a DoD environment, users are not generally trusted not to disclose sensitive data. The whole DoD security structure is designed to prevent sensitive data disclosure, even by a malicious user.

In a commercial environment, however, the view of security may be quite different. In a commercial development environment, such as the Digital environment for example, the users are generally trusted not to disclose sensitive data. The primary worry is the theft or destruction of data by a malicious penetration. Other types of commercial environments will have different security requirements.

The differences between the above two access control models clearly the more relaxed security controls of a commercial development environment. Two major access restrictions have been removed from the DoD model in creating the commercial model. Data sharing and ease of use are considerably enhanced using this model as compared to the more restrictive DoD model.

The addition of an integrity model for access control reflects the fact that data is a valuable resource to a company, and must be protected from destruction or corruption, as well as being protected from disclosure.

5.7 Related Reading

[1] *Department of Defense Trusted Computer System Evaluation Criteria,–*, DOD 5200.28.STD, December 1985.

[2] *Building a Secure Computer System*, Gasser, Van Nostrand Reinhold Co., 1988.

[3] *Distributed System Security Architecture Requirements*, Gasser, Goldstein, Kaufman, Lampson, Digital, Mar 1989.

[4] *Preliminary ABA Architecture Overview [Bean], rev 0.5*

[5] *OZIX Software Development Procedures*, Digital, Aug 1989.

CHAPTER 6

INTERNATIONALIZATION

6.1 Introduction

This chapter explains OZIX goals for internationalization and the design model used for OZIX software, and includes descriptions of individual OZIX international components. The purpose of this chapter is to provide a frame of reference from which to examine the design documentation of individual OZIX components.

6.1.1 OZIX Description

OZIX is an international product that provides a superior implementation of open systems software. The OZIX design takes into account requirements for compliance to open systems standards as well as requirements for localization and international capabilities.

All OZIX components facilitate translation by structuring program user interfaces and message text separately from functional code and by using multilingual messaging and language switching facilities. The OZIX base system software uses compound string technology to support multilingual software, and is code-set independent to the level of terminal services. Compound string technology is also incorporated into the OZIX file system, libraries and programming tools.

6.1.2 Terminology Definition

A definition of terminology is appropriate to begin this chapter. *Internationalization* is the process of developing an international product and delivering that product into worldwide markets. Internationalization incorporates two concepts—*localization* and *international capabilities*.

Localization is the process of adapting an international product to suit the language, conventions, and market requirements of a particular *locale* or local environment. Locale-specific conventions include cultural data representations such as radix and currency symbols, and formats for date, time, and calendar. Also dependent on locale are conventions for collating and sorting, keyboard mappings, character sets and character code sets, and conversion functions. *Translation*, which is the rendering of information presented in one natural language into another, is part of localization. Localization does not change the functionality of the international product.

International capabilities are those functions of an international product that support the languages and conventions of more than one locale. International products may be *localizable* or *multilingual*. Localizable software is software that can be modified to suit particular locales, while multilingual software has one version of software supporting multiple locales at the same time.

6.2 Goals

In concert with Digital's goal of providing worldwide software products, OZIX V1.0 is engineered with clear and pervasive internationalization goals. These goals may be classified as either design goals or functional goals. The functional goals are the following:

- Support for easily localizable applications
- Support for multilingual applications
- Good run-time performance
- Object level code compatibility with existing ULTRIX (RISC) applications
- Language and culture neutral base system components

The following are design goals in OZIX:

- Straightforward requirements for packaging, installation, and system management with respect to localized products
- Limiting locale-specific processing of internationalized data to the highest layer of software possible, typically at the presentation and user interface layers
- Logical decoupling of components such that localization can take place on a component by component basis to suit the local market requirements

The OZIX base system itself does not need to be multilingual, but it is important that the base system not preclude support for the execution of multilingual applications. In addition, not all base components must be localizable. For example, it may not be productive or possible to localize the console output produced during system crash recovery.

6.3 Design Model

The OZIX design for international support is a superset of several prior efforts to solve the general problem of creating international software. However, the requirement for OZIX to be an open system strictly tempers its design for internationalization, because OZIX must provide standard programming interfaces for traditional UNIX applications and interoperate with existing UNIX systems. The following sections explore the problems for which OZIX supplies solutions, and present the elements of OZIX internationalization with respect to previous and current design efforts.

6.3.1 Problem Statement

While American and European language versions of new software products are generally delivered to the local markets in a timely fashion, current engineering practices and projects do not produce products that can be easily localized for Far Eastern and Middle Eastern markets. Instead, additional engineering is required to build a product suitable for localization. This reengineering effort may include redesign of the product to handle additional character sets and screen display requirements. Reengineering, normally performed by local engineering groups in Japan, Hong Kong, Israel, and by International Engineering Development (IED) in Reading, adds to the cost of the localized product and seriously delays introduction of the localized product into the marketplace.

Digital is aggressively working to solve the problem of producing not only translated components, but also multilingual systems. For instance, *DEC STD 066-3* identifies the countries currently considered to be Digital strategic markets, and it is a corporate goal that new products be capable of simultaneous shipment to these strategic markets.

Digital is not alone in its effort to define solutions for internationalization problems. Various standards bodies are also attacking the problem. The American National Standards Committee X3J11 standard for the C Programming Language (Draft ANSI X3.159) includes a number of library functions that modify their behavior according to locale. ANSI C also defines a number of multibyte functions and an additional function for manipulating monetary values. Open Software Foundation (OSF) members are in the process of specifying the internationalization requirements for OSF/1 Operating System Component (OSC). The major thrust, however, has been in the X/OPEN Group.

6.3.1.1 X/OPEN Internationalization

The X/OPEN Native Language System (NLS) was first published in the X/OPEN Portability Guide (XPG-2), was refined in XPG-3, and is being considered for revisions in XPG-4. NLS defines facilities for the development of internationalized applications that use 8-bit coded character sets. NLS provides the following facilities:

- Message catalogues, to allow program messages to be separated from the program logic, translated into different native languages and retrieved by the program at the time of execution
- An announcement mechanism, whereby a locale appropriate to each user can be identified to applications at the time of execution
- Internationalized C library functions, to provide a facility for locale specific processing
- A set of library functions, to allow the program to dynamically determine data specific to a culture or language
- Regular expressions, to extend the standard provisions by providing for the specification of locale classes, accented base characters, and multicharacter collating elements
- A set of standard commands, to provide 8-bit transparency for the processing of data and the name of file storage objects

Under consideration for XPG-4 is the question of general multibyte character support. Proposals have been advanced to require full internationalization of the command set, including multibyte handling. Digital has advanced proposals for handling multidirectional text and for providing locale as an explicit argument to NLS operations.

OZIX is XPG compliant. With regard to internationalization, OZIX contains message catalogues that are compatible with NLS, and provides the NLS facilities listed above.

6.3.1.2 Multiple Octet Character Set

The Multiple Octet Character Set (MOCS) is a proposed standard for a worldwide character set, and is currently under consideration in ISO-IEC JTC1/SC2 as ISO/IEC JTC1 10646 DP. MOCS proposes mapping the characters of all existing character sets into a single character space populated with four-*octet* characters (an octet consists of eight bits). Digital is actively supporting the MOCS effort, which is expected to become a published standard by 1992.

MOCS is a character set definition that extends the concept of 1-byte character encoding arranged as a linear array to four-octet characters. MOCS solves the problems arising from mixed character sets by containing all characters in a single character set. MOCS proposes a 1,2,3 and 4-octet usage, with the possibility of setting the usage form via a self-announcement mechanism. The default is equivalent to the 8-bit ISO Latin-1 character set to provide backwards compatibility.

MOCS does not solve the problem of how to handle character strings that contain characters of different writing directions. For instance, right-to-left text may contain embedded left-to-right elements. When only MOCS is used for strings containing mixed writing directions, the writing must be derived implicitly from the characters themselves. Implicit derivation of writing direction, unfortunately, does not always lead to an unambiguous rendering of the text string.

OZIX provides library routines to perform conversions between MOCS and other character sets. OZIX uses strings of fully expanded MOCS characters (four octets) to store identifiers for such things as file names and device names. The identifier strings are not simple, but rather are of compound string type.

6.3.1.3 Compound String

A *compound string* is an opaque string type used to represent both the character values in string data and the attribute information needed to correctly process the string data. Compound strings are based on the Digital Data Interchange Format (DDIF) and Digital Data Interchange Syntax (DDIS), which are based on the ISO ASN.1 notation.

The attributes recorded in compound strings are mainly required to correctly render the characters of the string for human presentation. For example, attributes include specification of:

- Character set
- Language
- Writing direction

Compound strings also allow for string data of mixed character sets, language, and writing direction. These attributes are the minimum required for full internationalization support of languages such as Hebrew, Chinese, Korean, Japanese, and so on. The compound string format is easily extensible without change to the application program interface, allowing for the future addition of alternate representations of the string, including voice, graphics, and so on.

A compound string is not a simple linear array of characters; rather, it is a nested hierarchy of string segments and requires library support to be manipulated in all but the simplest manner. It is therefore potentially more costly to process compound strings (in terms of processor cycles and storage requirements) than either MOCS or most other simple character set-based strings. However, no model based on character sets alone provides the flexibility or future extensibility offered by compound strings.

Compound strings are supported in the DECwindows Toolkit as the sole interface to text string arguments, and are used in the declarations of text string resources in UIL widget definitions. They have been proposed to X/OPEN I18N Working Group as the mechanism to handle mixed writing directions.[1]

OZIX stores file storage object identifiers such as file names or directory names in compound strings containing four-octet MOCS characters. In addition, OZIX supplies a library of compound string entry points to provide standard I/O services plus additional library routines to manipulate compound strings or perform conversion between compound strings and simple strings.

[1] See the paper *Proposal For Mixed Writing Directions Support* by Mike Feldman, IED (June 5, 1989).

6.3.1.4 Producing International Products (PIP) International Product Model

The OZIX product model follows the guidelines for creating international software products as described in the *Producing International Products (PIP) Reference Set*. The PIP divides its international product model into four components:

- International Base Component (A Component)
- User Interface Component (B Component)
- Market-Specific Component (C Component)
- Country-Specific Information Component (D Component)

The A Component is the invariant part of OZIX, and can be sold worldwide without modification. Major elements of the A Component in OZIX are the base system, the file system, the messaging facility, and parts of the terminal services.

The B Component is the language and text processing component of OZIX, and must be localized to the language and cultural requirements of a specific local market. Major elements of the B Component in OZIX include message text, language specific processing in the terminal services, online help, documentation, and the user interface to OZIX components such as system management or diagnostics.

The C Component is an element added to the A Component to meet special requirements of a specific market, such as specialized text processing software for Japanese text. The C component extends the A Component without requiring any changes to the A Component.

The D Component is a set of mandatory documentation, packaging and labeling, warranty, support and product description information added to meet all regulations required to sell OZIX in a specified country. The D component places no special requirements on the design of the other components.

6.3.1.5 Unified Text Representation (UTR)

The Unified Text Representation (UTR) architecture has been proposed by Ron Brender as an approach to handling multiple character sets in Digital's software systems.[2] The main assertions of the UTR architecture are:

- MOCS is used as the primary character set throughout the system
- For files that are simple text, a file attribute is used to specify the text representation used in each file
- A file processing attribute is used by a program when opening a file to specify the representation wanted for processing
- Text is automatically converted (in both directions) between the two representations when they are different

OZIX extends UTR by using compound strings for system identifiers. The notion that pure text files can be tagged according to the character data contained within them is recognized in the OZIX system design and utilized by OZIX libraries to perform limited, automatic text conversions during file access between compound string files and 8-bit character data files.

[2] See the paper *Unified Text Representation (UTR): A System Overview* by Ron Brender, ABSS, (August 18, 1989).

In addition to UTR, a number of engineers from Asian Base Systems Software (ABSS) and IED have been working on a model for internationalized software, and have published their findings.[3] OZIX follows their recommended model, which is essentially UTR extended to use compound strings, with MOCS as one of the supported character sets.

6.4 OZIX International Components

This section describes the technology present in OZIX components to support execution of both localizable software and multilingual software.

6.4.1 Commands

An international command is a user level program that functions correctly regardless of the user's natural language. At a minimum all OZIX commands provide 8-bit transparency for the processing of data and the names of file storage objects such as file names or directory names. In addition, all OZIX commands use international message and language switching facilities, thus accommodating easy translation by local engineering groups. Some selected OZIX commands are fully internationalized and have the capacity to process data and file storage objects presented in character sets other than the ISO 8859-1 character set (8-bit Latin-1). OZIX internationalized commands use the OZIX compound string library routines to achieve this language neutrality.

\The selection of commands to internationalize is incomplete at this time. Selection is a joint decision process between marketing and engineering groups.\

6.4.2 Libraries

Internationalization support in OZIX libraries is contained in standard C libraries, compound string library extensions, and the compound string I/O library.

OZIX supplies a set of standard C libraries to provide string handling routines to deal with string parameters made up of 8-bit characters. Existing C applications link against the OZIX standard C libraries and enjoy complete compatibility.

OZIX extends the standard C library with an additional set of library routines that correspond to standard C library routines dealing with system identifiers. The compound string library extensions are used by new international applications that wish to be language neutral in expressing file names, directory names, user names, etc. Compound string library extensions depend on underlying operating system support for handling identifiers presented in compound strings.

Finally, OZIX provides a library of compound string I/O and utility routines that are used by new international applications. Utility routines manipulate compound strings and do conversions between compound strings and simple character strings. Compound string I/O routines parallel the standard I/O routines, except that the strings manipulated or processed through input and output are of compound string type rather than simple strings. The compound string I/O library is designed so that its routines are available to all languages supported in OZIX. It is a long-term goal that the compound string I/O library can be ported to other open systems.

Automatic conversion between compound string text and 8-bit text may occur in OZIX libraries. The key to library conversions is the concept of a data attribute to describe the character set contents of OZIX text files. For example, if an application uses the standard C library to access a text file that is tagged as compound string, the standard C library routines automatically convert compound string text data to eight-bit text data for delivery to the application. Likewise, if an application uses

[3] See the paper *Digital's Text Model for the Future* by Jürgen Bettels, Ron Brender, Tim Greenwood and Jim Saunders, August 22, 1989.

the compound string I/O library to access an 8-bit text file, the compound string library routines automatically convert 8-bit text data to compound string text data for delivery to the application.

Please note that no other types of conversion take place other than these two examples. In addition, file data attributes that describe character set contents may be examined or modified through a set of existing utilities that have been extended to understand file data types.

6.4.3 Terminal Services

The OZIX terminal subsystem provides support for multiple character sets and languages. The terminal subsystem is structured in accordance with the PIP A-B-C-D model, and allows for the inclusion of additional locale-specific character set handling without modification of the basic terminal subsystem code or data structures. To facilitate easy exchange of character data, the terminal subsystem uses the four-octet MOCS format internally for some of its character data buffers.

The primary interface to the terminal subsystem for an existing application is the OZIX OSF API. The terminal subsystem supports POSIX style functions such as *read* or *write*. In addition, the terminal subsystem supports communication of character data (as opposed to byte strings) through the OZIX compound string library interface.

6.4.4 Base System

The OZIX base system components are designed to provide language-neutral support for applications. Base system components are structured in accordance with the PIP A-B-C-D model by isolating message text to message catalogs, etc.

Base system components use four-octet MOCS characters within compound strings as the format for storing file storage object names such as file names, directory names, or device names, and for other system identifiers such as user names.

6.4.5 Messaging

The OZIX messaging facility is responsible for defining condition values and managing the association between a condition value and some short descriptive text (*message*). Condition values, system wide in scope, are associated with message text files that are created through the messaging facility. The data stored in message text files is retrieved at the time of execution through the use of message text routines furnished by the messaging facility.

In concert with open system requirements, the OZIX messaging facility supports open system message catalogs as defined in X/OPEN NLS. In addition, the messaging facility is multilingual; that is, it has the capability of handling multiple character sets and of supporting multiple concurrent locales. Pervasive use of the messaging facility throughout OZIX facilitates translation of components by separating message text from functional code.

6.4.6 Programming Tools

The OSG C compiler that is bundled for shipment with OZIX provides support for the development of international applications. The C compiler provides multibyte handling in source code comments, strings literals, and character constants. In addition, the C compiler provides wide character handling. It is the goal of the OSG C compiler to provide the support necessary for OZIX to be in compliance with the latest X/OPEN branding requirements.

Compound string support is provided in the OZIX run-time libraries. All development tools, such as the compiler, loader, and debugger, also have compound string handling capability.

6.4.7 File System

The OZIX file name subsystem is responsible for maintaining the hierarchical file namespace. The file name subsystem provides path traversal services and directory services to its clients.

The file name subsystem uses four-octet MOCS characters within compound strings as the format for storing file names, as well as other types of strings. As with other basic OZIX components, the file name subsystem is designed to provide language-neutral support for applications, and is structured in accordance with the PIP A-B-C-D model by isolating message text to message catalogs, and so on.

A file attribute to describe the character set contents of a file is maintained for each file. Each system maintains a default to be used for creating new files, when none is explicitly specified for file creation.

6.5 Additional Internationalization Support

Technology alone cannot guarantee that OZIX will be a successful international product. Other support required for success in the international market is described in this section.

6.5.1 Documentation

The OZIX documentation set is designed to complement the international OZIX product. In order to optimize translation to other languages, the entire information set is organized in accordance with the guidelines described in the PIP and in the *Planning for Translation* document. In recognition of the needs of different kinds of users, the information set is structured in a modular and hierarchical fashion to allow maximum translation flexibility. Local Engineering Groups may translate only selected information and structure that information as required to meet the needs of the local language and culture, without rewriting the original user information. Finally, stressing commonality of information across the entire ULTRIX family lessens redundancy in translation efforts.

Documentation tools also play a supporting role in the success of the international OZIX product. Online documentation tools for OZIX support the creation and display of European, Asian, and Semitic languages as well as mixed writing direction. Online documentation tools also support concurrent multiple languages, where the primary, secondary, and default language can be specified.

6.5.2 OZIX Release Strategy

A key part of the OZIX international product strategy is the configuration and coordination effort involved in creating the OZIX release kit. Three types of components create an OZIX system tailored to local market requirements:

- A common core system developed by the OZIX product team, combined with international Digital bundled and layered products and third party products.
- Country and language specific components, such as translated messages, language specific collating sequences, or a localized spelling checker library.
- Country-specific added value, such as tools tailored to a specific language or method of presentation.

At system installation, an OZIX kit localized for a specific market may be augmented by other country and language specific components. This augmentation allows a customer whose native language is Japanese, for instance, to produce applications suitable for other Far Eastern markets.

6.5.3 Training, Support and Service

Three other areas provide internationalization support to make OZIX a successful international product: training, support, and service. All three areas deal with the multilingual environment. The training curriculum for OZIX uses a variety of format types to best meet the learning and delivery needs of customers throughout the world. In order to support systems containing more than one local environment, services are developing a strategy for support of worldwide products and working to ensure that service personnel have access to appropriate resources. Finally, support personnel will provide expertise to help the customer develop international products or applications.

CHAPTER 7

BASE SYSTEM ARCHITECTURE

7.1 Introduction

The OZIX Base System Architecture defines the functional and conceptual primitives used to design the OZIX operating system.

Instead of defining a system consisting of a single, large, monolithic kernel, the architecture defines individual modules called *subsystems*, supported by a tiny "kernel" called the *nub*. Subsystems are the "building blocks" of the OZIX design.

A subsystem is a body of instructions and read, or read/write, data that provides a set of functions and data abstractions. Each subsystem provides functions that can be invoked by other subsystems in the OZIX operating system, which includes user applications. User applications themselves are implemented using subsystems; although from the user's perspective, they behave exactly like POSIX user processes.

7.2 OZIX Base System Architecture Goal

The goal of the OZIX base system architecture is to foster a crisply defined modular system design that achieves the aggressive security, integrity, and performance goals of the OZIX operating system.

7.3 Base System Architecture Structure

The OZIX base system architecture defines two types of functional components: subsystems and the nub. Since an operating system cannot be defined by function alone, the virtual memory environment and execution of subsystem functions are defined by two *models*: the *execution* model and the *virtual memory* model.

The two types of functional components and the two conceptual models that make up the architecture are briefly summarized below, then discussed in detail in the following sections.

The two functional components of the architecture are:

- Subsystems

  Subsystems are the modular functional units of the base system architecture. Each subsystem serves a specific role, such as file system, memory management, or device driver support. Subsystems provide the basis for controlled isolation in the operating system design and implementation. This separation of subsystems is in turn the basis for the integrity and protection of the entire system.

  Subsystems are also the basis for extending and evolving the OZIX operating system in the future. Whenever a new OZIX system capability is to be made available, a new subsystem can be incorporated, thus extending the operating system itself.

- The Nub

  As its name is meant to imply, the nub represents a very small part of the overall system code. The nub contains the primitive functions necessary to efficiently execute the code contained in subsystems. As a result, the nub contains most of the code that is dependent on the specific processor and hardware virtual memory architectures.

The execution and virtual memory models define how subsystems and the nub relate to physical memory and the processors to perform useful work.

- Executor Model

  The executor model defines how user and system computational tasks are described and supported by the OZIX subsystems.

  An *executor* consists of a family of threads executing code to perform computational tasks which may be implemented in one or more subsystems. In OZIX, the traditional "user process" is simply an executor originating in an application subsystem.

  The security and accounting identity of an executor is established at the time of its creation. An executor's threads maintain this same security and accounting identity as they pass through the various subsystems.

- Virtual Memory Model

  The virtual memory resources used by executors are described by the virtual memory model. The virtual memory model presents memory in terms of *memory segments*, which represent subsystem code and executor data.

  The primary goal of the virtual memory model is to allow the precise specification of an executor's access to, and use of, virtual memory. The virtual memory accessible by an executor while it is executing code in a subsystem represents a unique view of memory. This subsystem/executor-specific view of memory is referred to as a *subsystem execution context*, or *SEC*.

  The virtual memory model isolates the hardware-dependent CPU and memory management details from the subsystem designer. This helps make the entire system less dependent on specific CPU architectures.

7.4 OZIX Base System Architecture Functional Modules

All of the code in the OZIX system is executed in either a subsystem or in the nub. The vast majority of code is executed in subsystems, as the nub contains only the minimum functions necessary to support the virtual memory environment and execution of code within a subsystem.

Common procedures used by many subsystems may be implemented by code libraries. When such library procedures are called by an executor in a given subsystem, they execute in the same SEC as the subsystem code that called them. For this reason, a library procedure is considered part of a subsystem's code from the perspective of the architecture.

Subsystems and the nub are described in Section 7.4.1 and Section 7.4.2, respectively.

7.4.1 OZIX Subsystems

Each OZIX subsystem supports a documented interface through which specific functions and data structures can be accessed by executors from other subsystems. OZIX subsystems can be thought of as object oriented in the sense that they can export protected objects, as well as operations that can be performed on those objects.

A subsystem can be described from two perspectives:

- External Perspective - The subsystem as viewed by the caller
- Internal Perspective - The resources used by the subsystem on behalf of the caller

The external perspective of a subsystem is nothing more than its procedural interface; that is, the functions it can perform on behalf of executors. An executor in one subsystem can invoke a procedure in another subsystem by issuing a *subsystem procedure call*. Subsystem procedures are discussed in Section 7.4.1.1.

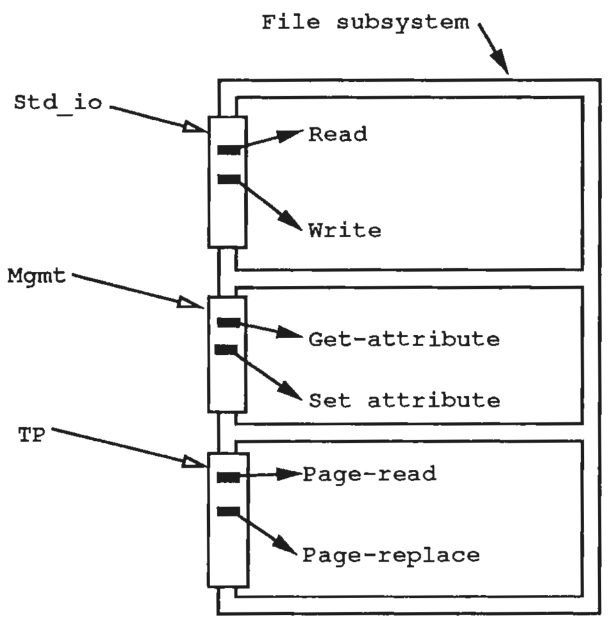
The internal perspective of a subsystem is the execution environment available to the executor while executing a subsystem's functions. This view of memory is the subsystem execution context, which is explained in Section 7.4.1.2.

7.4.1.1 Subsystem Procedures

Access to a subsystem's procedures are controlled on a per-executor basis. The entry points to a subsystem's procedures can be grouped into *packages*, which control access by a given executor to the group of subsystem entry points. By grouping subsystem entry points into packages, an executor, for example, may be granted access to all of the procedures in the "service" package of a subsystem, but denied access to the procedures in the "management" package.

Subsystem procedures are named hierarchically. Any subsystem procedure in OZIX has a name of the form *Subsystem_name.Package_name.Procedure_name*. This way a common *File_io* package can be defined, even though many subsystems may implement that package.

Figure 13: A simple subsystem



A functional view of a subsystem is illustrated in Figure 13. This simple example shows the "File" subsystem implementing three packages of two procedures each. The "Std_io" package implements support for the standard OSF "Read" and "Write" functions. The "Mgmt" package implements the standard OZIX system management "Get-attribute" and "Set-attribute" procedures. The "TP" package implements value-added "Page-read" and "Page-write" procedures for use by transaction-processing database subsystems.

7.4.1.1.1 Subsystem Procedure Calls (SPC)

Subsystem procedure calls, or SPCs, are used by an executor in one subsystem to invoke a specific package entry point, or procedure, in another subsystem. From the perspective of the caller, a subsystem procedure call is identical to a normal procedure or system call. In order to better understand an SPC, first consider the essential steps involved in a normal procedure call:

1. Code in the caller assembles the call parameters, or argument list, according to the calling standard. The calling standard typically specifies registers for some number of arguments, and stack local storage for the remainder.
2. Some form of linked jump is performed, which saves return information and transfers control to the call site, or called procedure.
3. Code at the call site will typically save registers used by the called procedure, possibly perform other bookkeeping or optimization operations, and then begin execution of the procedure body.
4. Control is returned back to the caller through the saved jump linkage.

An SPC is nearly identical. The basic steps are:

1. Assemble call parameters in the caller in exactly the same manner as the normal procedure call above.
2. Execute a protected transition to the called procedure while changing to the SEC of that procedure's subsystem.

3. Execute the procedure code at the called site in the same manner as a normal procedure call.
4. Execute a protected transition back to the caller, restoring the caller's SEC.

7.4.1.1.2 Gates

The difference between a normal procedure call and an SPC is the linkage mechanism used to transfer control from the caller to the called procedure, and back to the caller. This mechanism, used in Steps 2 and 4 above, is referred to as a *gate*. The protected transition implemented by a gate is referred to as a *gate crossing*.

The gate crossing mechanism is designed to operate with as little overhead as possible. In particular, no access or security checks are performed by the gate mechanism itself, which is only responsible for executing a protected transition from one SEC to another SEC. Any necessary access checks are made when an SEC is constructed, or possibly deferred until the first reference is made to some memory object in the SEC.

Likewise, the gate does nothing with the procedure call argument list beyond making the arguments available in the called procedure's SEC. In particular, the gate does not track down indirect references, such as those represented by pointers. This ensures that the gate is kept simple enough to provide maximum performance.

7.4.1.1.3 Implicit and Explicit Subsystem Procedure Calls

Subsystems are dynamically loaded into the OZIX system. As a consequence, calls to subsystem procedures are dynamically resolved when they are initially executed, much like calls to shareable library routines are resolved by autoloading code when first called.

An SPC can be made either implicitly or explicitly. An implicit SPC is coded in the same manner as a call to an external routine. Compiler, linker, and autoload routines create the illusion of a normal procedure call using shared library mechanisms.

An explicit SPC uses the symbolic subsystem procedure name resolution and gate crossing mechanisms directly when subsystem calls can only be resolved at runtime by the calling subsystem. It is comparable to calling a function through a pointer in traditional systems. Examples of subsystems that require this dynamic call resolution capability are:

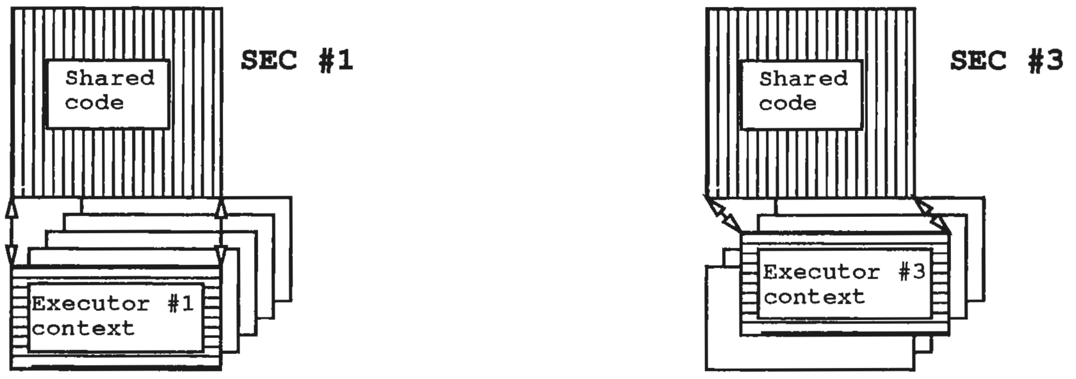
- A subsystem which dynamically resolves references to file and device subsystems as part of pathname traversal
- Network subsystems, which must dynamically resolve references to various transport and routing subsystems for a specified protocol stack

7.4.1.2 Subsystem Execution Context

A subsystem execution context is the set of memory segments accessible to an executor within a given subsystem. Figure 14 illustrates a number of key points about subsystem execution context.

Both figures in the illustration show a subsystem with shared code and executor-specific context. In the left hand figure, the SEC for executor #1 is highlighted. In the right hand figure, the SEC for executor #3 is shown. In both cases, the same shared code segment is accessible. The difference is simply which executor-specific segment is mapped.

Figure 14: Subsystem Execution Context



There are many design possibilities between the strictly executor-specific memory segments and completely shared segments illustrated. The degree of sharing is determined by the subsystem designers and controlled by the access control associated with the logical memory segment.

7.4.2 OZIX Nub

The nub provides the primitive functions required to execute subsystem code. The functions provided by the nub are those that require execution of privileged hardware instructions, access to physical memory, visibility across address spaces, or those functions that occur in special hardware states. Nub functions include thread scheduling and synchronization, CPU management, subsystem gate crossing, interrupt dispatching, and fault detection primitives.

The OZIX nub is a small piece of software, since most of the traditional kernel-level functions are implemented by the subsystems. In order to simplify the task of migrating to different hardware platforms, the nub is structured as two subcomponents, one hardware-dependent and the other hardware-independent.

The nub code executes in a hardware protected mode, such as kernel mode. The nub must execute in a protected execution context in order to have the capability to make the protected transitions between subsystems, which execute in separate virtual contexts.

The nub supports fault detection and recovery by detecting both hardware and software faults and initiating global fault recovery through the fault recovery subsystem.

7.5 OZIX Base System Architecture Conceptual Models

As discussed earlier, subsystems and the nub operate within the framework of two basic architectural concepts: the executor model, the virtual memory model. Both of these models are discussed in detail in the following sections.

7.5.1 Executor Model

An executor, loosely, is a family of threads working together on a common task. A thread represents a machine state, an execution stack, and a security and accounting identity. More precisely, then, an executor is defined to be a set of threads that share the following attributes:

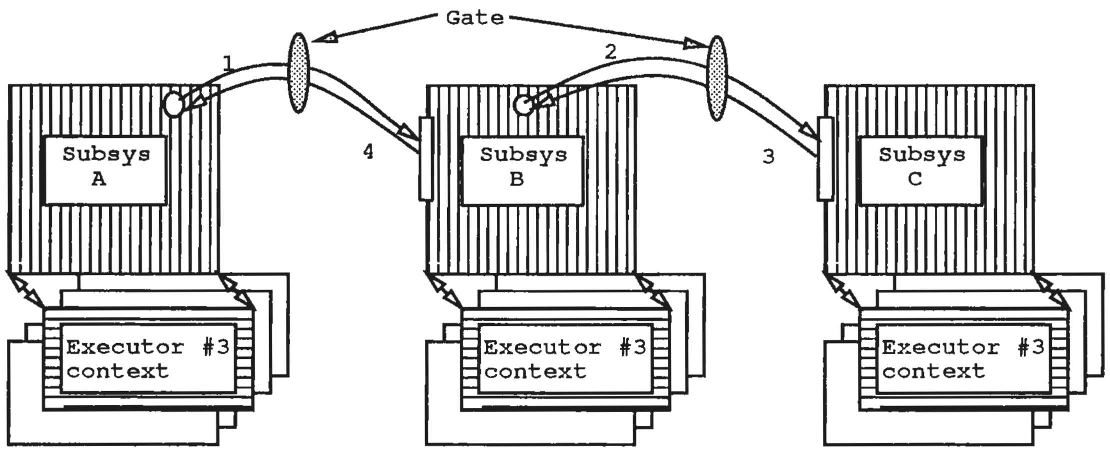
- User Identity
- Security and Integrity Labels

- Accounting ID
- Set of Subsystem Execution Contexts

Figure 15 illustrates a single executor, identified as "#3", making an SPC from subsystem "A" to subsystem "B", which in turn makes an SPC to subsystem "C", then returns. In this example, each subsystem execution context contains executor-specific context, as shown by the "Executor #3 context" memory segment in each subsystem. These executor-specific segments are private to each subsystem. Note, however, that it is the same segment (and contents) whenever any thread of executor #3 is in a given subsystem.

An executor executes "through" subsystems with a set of related and persistent subsystem execution contexts. The individual threads of a subsystem are scheduled for processor execution. The SPC gate mechanism itself does not cause any scheduling decisions to be made.

Figure 15: Executor SPC calls



There are many active, but distinct, executors. Some executors may represent customer applications, while others represent special system support functions. From an architectural standpoint, executors differ only in their task and one or more of the executor attributes listed above.

7.5.2 Virtual Memory Model

The OZIX virtual memory model builds an abstract virtual memory environment from the physical memory and virtual memory mapping features of the hardware. It is within this virtual memory environment that subsystem code executes. There are three layers of abstraction to the virtual memory model:

- Physical Memory
- Logical Memory Segments
- Subsystem Execution Context

The memory that is actually implemented by the hardware system is, of course, the *physical memory* available to the OZIX operating system. Abstract vectors of pages are known as *logical memory segments* (LM segments). LM segments describe the mapping of logical pages to physical pages. The dynamic reassignment of the limited number of physical pages to the much larger set of logical pages is a function of *physical memory management*.

A subsystem is composed of one or more logical memory segments containing executable code, and one or more logical memory segments containing data. The set of logical memory segments mapped to a specific executor while executing in a given subsystem is known as a *subsystem execution context*, or SEC.

Figure 16: OZIX Virtual Memory Model

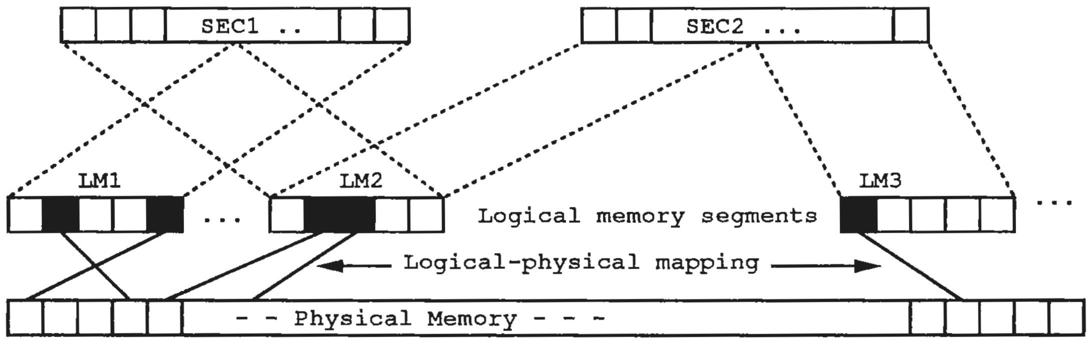


Figure 16 illustrates the layers of the virtual memory model. In this example, the black pages in the logical segments denote pages backed by physical memory. Three logical segments are mapped into two SECs. Logical segment "LM2" is mapped to both "SEC1" and "SEC2". This would be typical of a shared code segment.

7.5.2.1 Physical to Logical Memory Mapping

The lowest-level, hardware-independent, abstract representation of memory in the virtual memory model is the *logical memory segment*, or LM segment. An LM segment is a logical set of memory pages with access permission controlled by an ACL and label.

The pages in an LM segment are themselves mapped to physical memory resources in a hardware-dependent fashion. LM segments define one of the fundamental hardware dependent/independent boundaries in the OZIX system architecture. LM segments insulate memory managers from the idiosyncrasies of the particular hardware implementation represented by the OZIX physical memory management software. For example, in a paged machine, an LM segment may contain the mapping information and a page table.

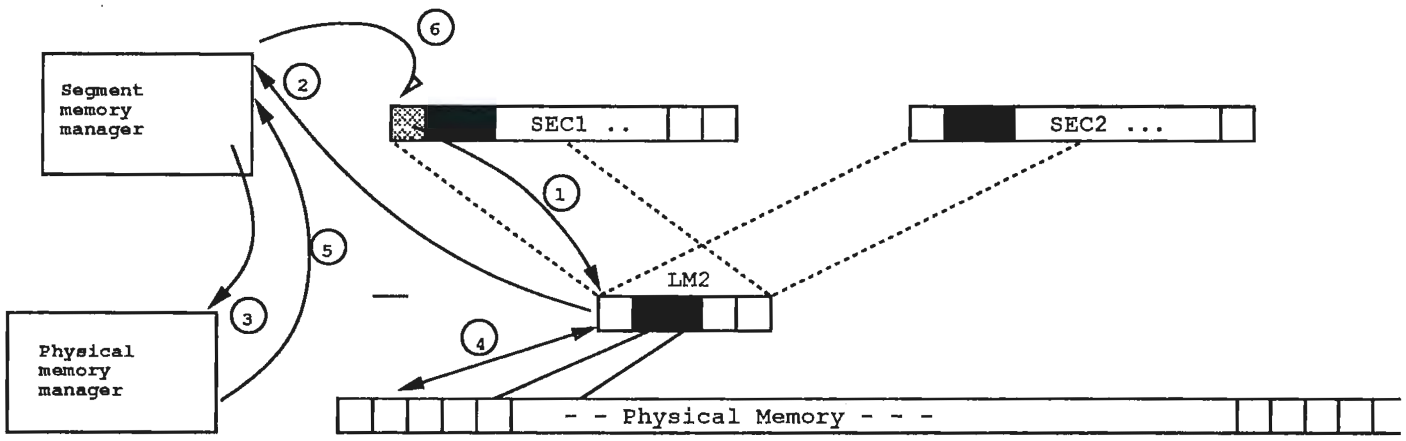
7.5.2.2 Physical Memory Management

OZIX physical memory management uses LM segments as the basis for physical memory resource accounting and management. The OZIX physical memory management software maintains the state of the LM segment mapping information. In a way, physical memory can be viewed as a "cache" of LM segment content. In general, the physical memory management software attempts to remove physical memory mappings from inactive logical segments so the physical memory can be reassigned to active logical segment pages.

The OZIX virtual memory model defines a very flexible and general mechanism. Simply stated, each LM segment has a *segment memory manager* associated with it. The segment manager for a given LM segment implements the "inpage" and "outpage" algorithms for the logical pages in that segment. This allows paging algorithms to be precisely tuned for different uses of virtual memory.

Another way to view this is that segment memory managers negotiate with the physical memory management software to add and delete physical memory for their LM segments. The segment memory manager makes requests to the physical memory manager for physical pages to satisfy page faults for segment pages, and the physical memory manager makes requests to the segment memory manager to reclaim inactive physical pages for use by the actively faulting segments.

Figure 17: Logical segment page fault resolution



The directed arcs in Figure 17 illustrate the simplified interaction of a segment memory manager and the physical memory manager to satisfy a page fault in SEC1.

1. The faulting page in SEC1 is traced to the appropiate page in logical memory segment LM2.
2. There is no physical memory mapping for the logical page in LM2, so a call is made to the segment memory manager for LM2.
3. The segment memory manager requests a physical page mapping for the faulting logical page.
4. The physical memory manager maps a physical page to the desired logical page.
5. Control returns to the segment memory manager with a valid virtual mapping for the faulting page. The segment memory manager must now initialize the page appropriately. For example, the segment memory manager might read a page from a file if it is implementing a "mapped file" type of segment.
6. After the segment memory manager has successfully initialized the page, it returns control to the virtual memory fault handler, which restarts the faulting instruction.

Figure 18: Physical memory page reclamation

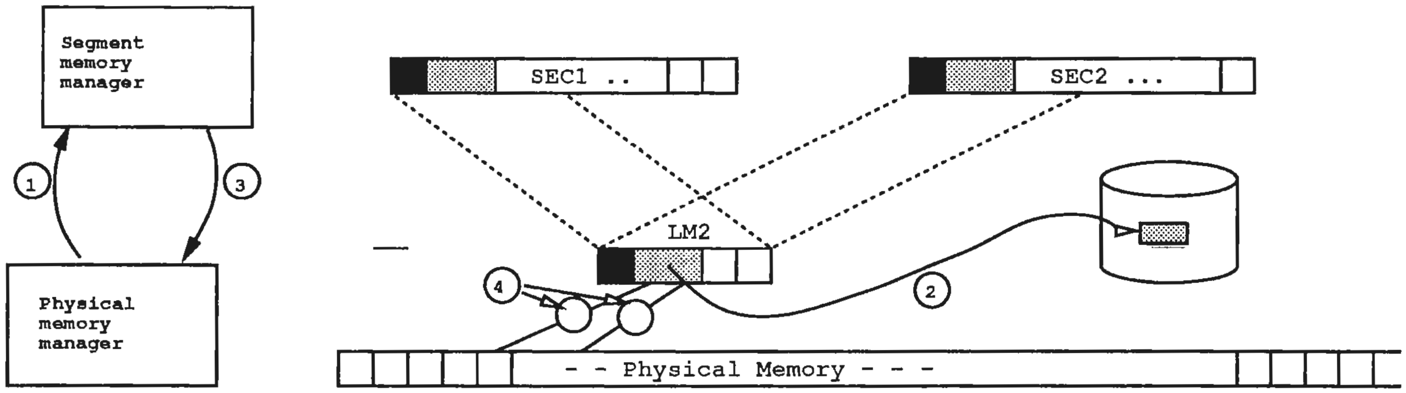


Figure 18 illustrates a simplified example of the physical memory manager reclaiming pages from a logical memory segment.

1. The physical memory manager has determined that it needs to reclaim two physically mapped pages from logical memory segment LM2. It calls the segment memory manager for LM2 and requests it to free up two pages.
2. The segment memory manager decides which two pages to unmap out of the three that are physically mapped. These are noted by the cross-hatch pattern in the figure. It writes them to mass storage so they can be recovered later.
3. After successfully saving the two pages on mass storage, the segment memory manager returns control to the physical memory manager.
4. The physical memory manager breaks the physical mapping for the logical pages specified by the segment memory manager.

The significant features of this memory management model are:

- There can be many different logical page management algorithms and policies beyond standard file paging. They can be tailored to satisfy specific requirements, such as a data buffer cache.
- Physical to logical management is simple and direct. When the physical memory manager decides it wants a page back from an LM segment, it gets it.

7.6 Summary of OZIX Base System Architecture Features

The OZIX base system architecture defines a truly modular system for designing and implementing an operating system. Key distinguishing features of this architecture are:

- Subsystems are used to implement essentially the entire system. Subsystems define independent, protected execution contexts, which provide strong data encapsulation capabilities to system designers. This is in marked contrast to typical commercial operating systems, in which the entire system operates in a very small (at best) number of protected execution contexts.
- A memory model which allows different memory management algorithms to efficiently compete for common physical memory resources.
- An execution model, which separates scheduling decisions from execution environment transitions, as well as defining the primitive unit of identification in the security model.

The OZIX base system architecture creates a *real* modular framework upon which the OZIX system is built. As such, it represents a significant step beyond the use of modular design techniques. It creates profound opportunities for elegant and innovative design solutions to the perennial operating system problems of reliability, security, integrity, extensibility, flexibility, and continuous operation.

CHAPTER 8

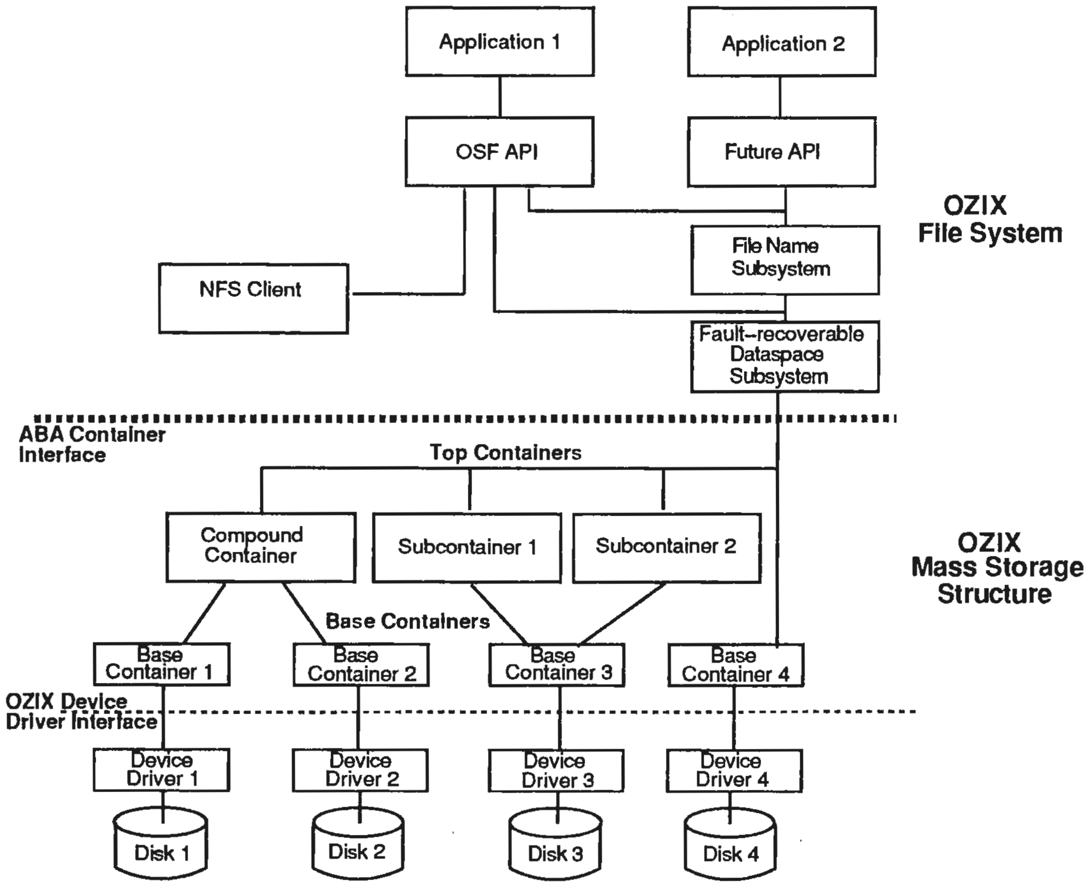
I/O SYSTEM

8.1 Introduction

OZIX I/O consists of a modern file system on top of a highly structured mass storage system. Both the file system and mass storage system designs are based on the emerging Digital attribute-based allocation (ABA) architecture.

Figure 19 illustrates the I/O components discussed in this chapter. The top portion of the I/O system is implemented by the file system, and the lower portion is implemented by the mass storage system. This chapter first introduces the ABA architecture, followed by separate discussions on the OZIX file system and mass storage system.

Figure 19: Overview of OZIX I/O System



8.1.1 Goals

The goals of the OZIX file system and mass storage designs are as follows:

File System Goals

The main goals of the OZIX file system design are to:

- Create a faster, more robust file system that is accessible through the POSIX 1003.1 file system interface. The file system design exploits transaction processing techniques to guarantee seven-days-a-week/twenty-four-hours-a-day (7/24) reliability, minimize response time, and optimize the updating of on-disk data structures.
- Design the OZIX file storage architecture to handle very large disk farms. This capability is made possible by developing the *attribute-based allocation* (ABA) concept into a workable architecture. The ABA architecture will initially be implemented on OZIX, and may be implemented on other operating systems in the future.
- Extend the POSIX interface by adding resource manager support services to support data base systems in a transaction processing environment.
- Support the OZIX B2 security design. Dataspaces are objects in the OZIX security system, and as such have sensitivity and integrity labels, and are subject to mandatory access and discretionary access controls.

- Support Hierarchical Storage Management (HSM) for better utilization of storage devices. Devices are organized in a hierarchy of faster and slower devices, and HSM migrates files between devices.
- Create a filename subsystem that can be called by multiple file system interfaces. Initially a POSIX file system interface will be developed, but additional file system interfaces—such as Apple® AFP, and MS-DOS SMB—could be developed.
- Provide a media failure recovery mechanism for fast and reliable dataspace recovery from media failures.

Mass Storage Goals

The major goals of the OZIX mass storage design are:

- Provide the basis for very high I/O throughput at very low latency. A set of ABA mechanisms support highly variable hardware topologies where available hardware capability can be applied where it is most needed.
- Support both DSA and SCSI devices. The design will also be able to support future storage technologies, interconnects and peripherals.
- Provide the capabilities to create storage abstractions, such as disk striping and shadowing, from the underlying storage hardware configuration and interconnects. Such storage abstractions are implemented as ABA abstractions on top of elemental devices.
- Support many different types of mass storage devices—including solid state devices (SSD), magnetic and optical disks, magnetic tapes, and CDROMs – by implementing drivers for the various device types.
- Provide for a robust I/O configuration management by implementing an I/O configuration manager based upon the Enterprise Management Architecture.
- Provide the capability of being extensible by designing the mass-storage components that takes full advantage of features provided by the OZIX Nub and other subsystems, like the memory management subsystem and the process subsystem. Mass-storage components will be designed to be adaptive so that the components operate optimally, within the bounds of the system resources.

8.2 Attribute-based Allocation Architecture

The overall goal of the ABA architecture is to make the specifics of allocating and managing storage media transparent to the file system or application. The ABA architecture is designed to separate file storage allocation from device location by presenting higher levels of software with an abstract view of mass storage.

ABA effectively manages a multi-level hierarchy of differing storage devices, according to a set of policies. Using ABA, it is possible for storage management systems to provide 7x24 service for very large file and database systems with minimal impact on running applications. Using ABA, it will be possible for storage systems to deliver average throughput approximating that of the fastest data storage device with average costs and capacity approximating that of the least expensive storage device.

The ABA architecture is made up of a *dataspace* layer and a *container* layer. Dataspaces are implemented by the file system; containers are used in the OZIX mass storage system.

Dataspaces

The ABA dataspace layer represents mass storage media as being made up of logical units called dataspaces. A dataspace is a virtually addressed storage space to which data is written and from which data is retrieved. A dataspace may represent part of a file, a file, or several files. Each dataspace is associated with specific attributes that define its storage requirements, size, access characteristics, and so on.

Containers

The container layer of ABA presents all mass storage devices as *containers*, which are logical representations of the devices. Containers representing physical devices are known as *base containers*. Base containers may be partitioned into *subcontainers*. Containers that represent virtual device abstractions required for such functions as striping and shadowing are known as *compound containers*. Compound containers can also be partitioned into subcontainers.

As dataspaces are associated with attributes, each container is associated with specific properties. For example, a base container would have properties that indicate details about the underlying device, such as its speed, size, reliability, and so on.

Dataspace Migration

Instead of allocating a specific location on a specific disk, dataspaces are allocated in a container having the properties best suited for the particular file. Dataspace migration can occur because a user changes a file's attributes, the system manager requests a migration, or because storage management detects discrepancies between the storage resources desired and those that are available.

OZIX implements numerous migration strategies, such as:

- Migrate dataspaces to cheaper containers.
- Migrate dataspaces to high performance containers at a particular time of day when heavy use is anticipated.
- Migrate dataspaces within the same container when they become fragmented, thus consolidating them in contiguous container blocks.

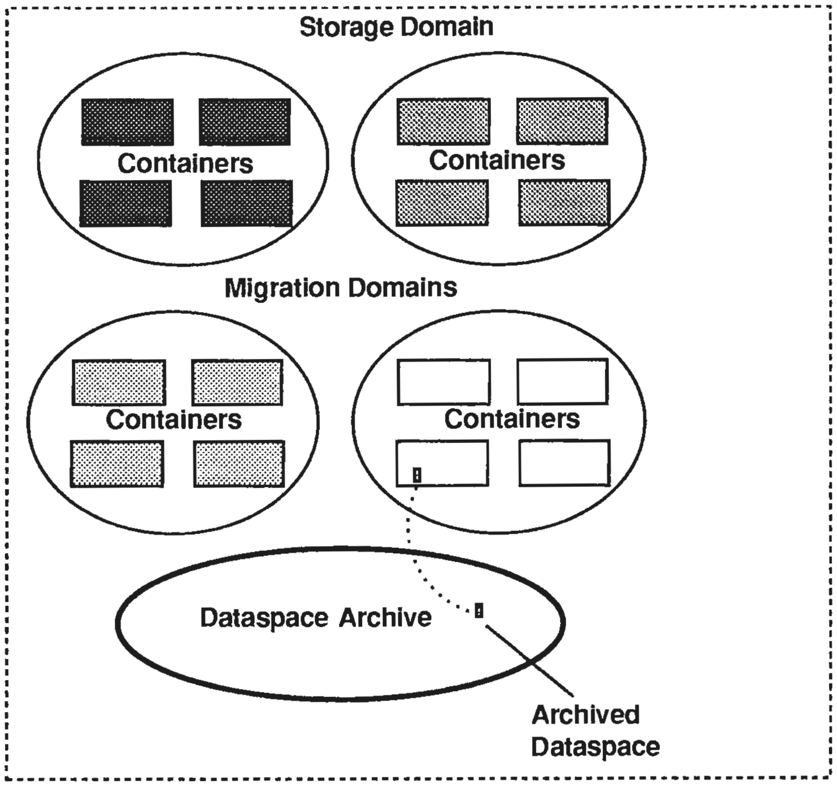
As illustrated in Figure 20 dataspaces migrate within a specified *migration domain*. Dataspaces may also be archived, in which case the dataspaces move out of their native migration domain into a *dataspace archive*. The dataspace archive is a repository for dataspaces that are not expected to be active for a long time. The entire storage domain of a system consists of one or more migration domains and one or more dataspace archives. Upon request, the dataspace from a dataspace archive can move back to its original migration domain.

8.3 File System

The OZIX file system is designed to provide quick and reliable access to files and data. Much of the design places special emphasis on fulfilling the needs of transaction processing (TP) resource managers, such as database systems. The functions used to support such resource managers provide the foundation for a superior general-purpose file system design and implementation.

The top portion of Figure 19 illustrates the OZIX file system. Note that only the OSF API is being implemented for V1 of OZIX.

Figure 20: ABA Storage, Migration, and Archival Domains



8.3.1 API Subsystems

The top level of the OZIX file system design is composed of API subsystems. These subsystems present a specific file interface to an application, such as the POSIX 1003.1 interface. OZIX is designed so that multiple APIs can be written to interact with the OZIX file system. Examples of APIs other than POSIX that could be developed are Apple AFP or MS-DOS SMB.

8.3.2 File Name Subsystem

The File Name Subsystem is the next layer of the OZIX file system. The file name subsystem presents a generic set of file and directory services that may be called by any API in OZIX. The File Name Subsystem provides directory and file services to the ABA-based files below the File Name Subsystem. Any API can, for example, open, read, write, and close an ABA-based file.

The File Name Subsystem also provides services to create mount points to mount objects other than local ABA files in the local directory hierarchy. These services, for example, allow remote directory trees that are accessed via NFS client services to exist in the local directory structure, and provide create and lookup routines for pipes and special files in the local directory structure.

8.3.3 Fault-recoverable Dataspace Subsystem

The Fault-recoverable Dataspace Subsystem implements the ABA architecture dataspace concept. This subsystem implements services to create and access dataspaces, to transfer data to and from dataspaces, and to manage dataspaces. This subsystem implements a buffer pool cache to provide read caching and optimized write-back strategies; the Fault-recoverable Dataspace Subsystem subsystem also provides dataspace logging and recovery services, as discussed in Section 8.3.4.

8.3.4 Features Provided by the File System

The subsystems making up the OZIX file system, combined, provide the following capabilities.

Hierarchical Storage Management

Hierarchical storage management (HSM) uses the ABA architecture to organize file data in a device hierarchy made up of faster and slower storage devices. At the top of this hierarchy there might be solid state disks, followed by on-line disks, high-density cartridge devices, and finally off-line magnetic tape devices. HSM moves file data around in the device hierarchy so that commonly accessed file data is stored on faster devices, and less frequently accessed file data is stored on slower devices.

Logging and Recovery

The Fault-recoverable Dataspace Subsystem provides system-wide logging and recovery services. These services support the logging needs of multiple log users, utilizing either a shared common log or independent logs. Users of logging and recovery include the file system, which logs updates to file metastructures, and select transaction processing applications, which log updates to important transaction data. In the event of a system crash, the log files are used to recover the system state at the moment it crashed.

Security

The OZIX file system supports discretionary access control list (ACL) security and non-discretionary secrecy and integrity controls on dataspaces.

Secrecy and integrity controls allow users to be segregated into different security levels and provide controls for restricting the access between those levels. Such controls are mandatory on B2 certified security systems.

Secrecy controls prevent users at lower levels from reading files maintained by users at higher levels and users at higher levels from writing files to lower levels.

Integrity controls keep a security breach that entered the system at a certain integrity level from moving to higher levels. Under these controls, users at lower levels are unable to write to higher levels, and users at higher levels are unable to read from lower levels. Such integrity controls represent significant added value for commercial systems.

8.3.5 Backup Strategy

OZIX provides two classes of backup technology to end-users. The first class consists of UNIX file backup utilities *tar* and *cpio* which provide standard mechanisms for file backup and data interchange among UNIX-based systems; there are not plans to support *dump/restore*. The main goal of this class is to provide compatibility with UNIX and UNIX standards such as X/OPEN and POSIX.

The second class provides Digital value-added backup and recovery features, which are able to handle the high-availability requirements of the large commercial systems which OZIX is intended to support. Since dataspaces and containers are the fundamental storage elements in OZIX, this class of backup is concerned with back up and recovery of dataspaces rather than files. Also, since the major goal is to provide recovery from media failures, the selection of the dataspaces to back up is driven by the containers in which the dataspaces reside. This is in contrast to the file system and directory oriented backups done with *tar* and *cpio*.

8.4 Mass Storage Subsystems

The OZIX mass-storage subsystems are designed to meet the requirements of transaction processing (TP) systems, which process hundreds to thousands of transactions per second. Systems that process such large volumes of transactions require a large number of mass-storage devices.

The multitude of storage devices required to handle multiple terabytes of data have a wide range of cost, reliability, availability and performance characteristics. The OZIX mass-storage strategy provides opportunities for implementing a hierarchical storage management system that can efficiently support such a diverse range of storage devices. The key element of OZIX I/O strategy is the implementation of Attribute Based Allocation (ABA) architecture. OZIX mass-storage implements the *container* aspect of the ABA architecture.

As discussed in the Security chapter, all system components that might be used by an unauthorized user or program to gain access to the system must be part of the trusted computing base (TCB). Since most I/O device drivers access data belonging to multiple users, as well as data at multiple sensitivity levels, they must be part of the TCB.

OZIX has a comprehensive system administration interface that conforms to the emerging Enterprise Management Architecture (EMA). Mass-storage devices are automatically managed by the I/O configuration management (IOCM) application discussed in Section 8.4.3.

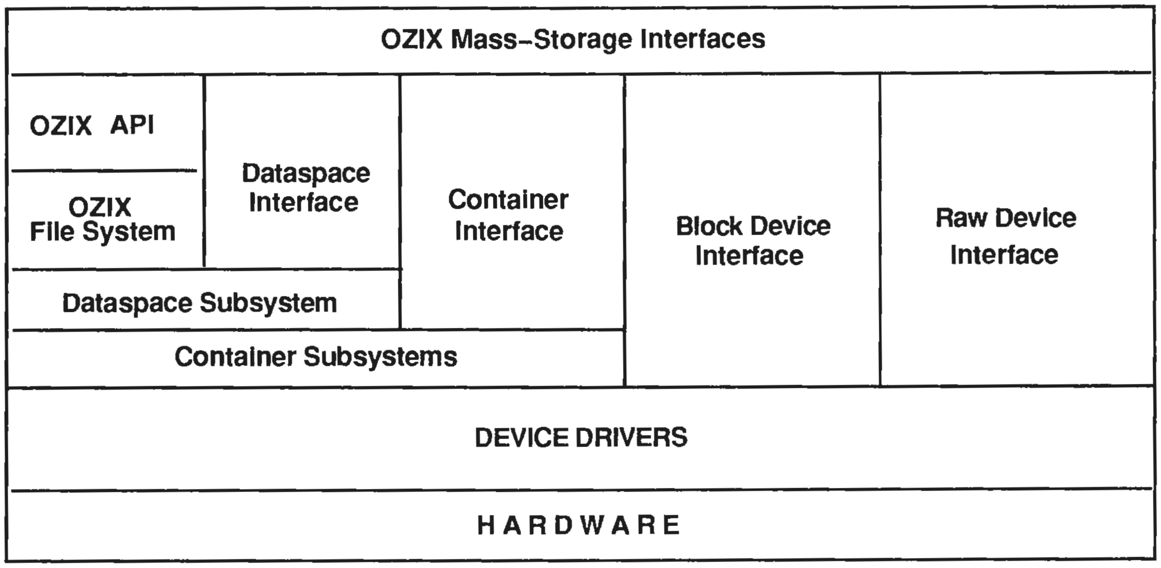
OZIX system will be booted by way of the code resident in a ROM or a console device, that is consistent across all operating systems using the hardware platform.

In OZIX, I/O operations can be targeted to mass-storage devices through any of the following interfaces:

- The Application Programming Interface (API)
- The Dataspace Interface
- The Container Interface
- The Character Device Interface
- The Block Device Interface

Figure 21 shows various I/O interfaces that are available through the mass-storage subsystems in OZIX.

Figure 21: OZIX Mass-storage I/O Interfaces



The API and dataspace interfaces are discussed in the the OSF Application Programming Interface Functional Specification, and the Fault-recoverable Dataspace Subsystem Functional Specification. As illustrated in Figure 21, the container subsystems, together with the device drivers and hardware provide support for the ABA container interface and both block and character device interfaces.

8.4.1 Mass Storage Components

Supporting a system that processes large numbers of transactions imposes several stringent requirements on the mass storage. The primary requirements are:

- Seven-days-a-week, twenty-four-hours-a-day (7x24) operation, while handling terabytes of data
- Very high I/O rates that facilitate transaction turnaround time of 2 seconds or less
- Very high data availability

As illustrated in the bottom portion of Figure 19, OZIX mass storage consists of two major components:

- Containers
- Device drivers

8.4.1.1 Structuring Containers

As discussed in Section 8.2, the dataspace layer of ABA views mass storage in terms of containers, which are ABA abstractions that represent physical devices. Each container is identified by a unique global name, a UID, and a set of properties. Container properties represent state information in the same manner as the EMA-defined attributes.

A single physical unit is represented as a base container. In OZIX V1, base containers only represent random access devices, such as magnetic disks; it is undetermined whether sequential access devices, such as magnetic tapes, are to be represented by containers in later releases.

Containers can be combined to form *compound containers*, which present multiple base containers as a single logical device to the dataspaces layer. For example, disk shadowing is implemented by combining two base containers, which are logical representations of, say, two magnetic disk devices, into a single shadowed compound container. To the dataspace layer, the shadowed container is viewed as offering higher availability than either of the two base containers from which it is built. The OZIX I/O strategy is to offer a range of features based on compound containers. The following compound containers are expected to be available at V1 FRS:

- Shadowed container—This container consists of two or more constituent containers. Data is replicated in each of the constituents. Replication of data increases data availability and enhances data reliability.
- Striping container—This container consists of two or more containers on which data is distributed in a manner that provides optimum throughput and low latency. Applications that need high data transfer rates can use this container to meet their transfer rate requirements.
- Linear catenation—This consists of two or more containers to form a large virtual device. The address spaces are simply catenated. This compound container is suitable for holding large files that do not require high transfer rates.

The following compound containers are expected after V1 FRS:

- Caching container—This is made up of two containers: one is a large, relatively slow backing store device (for example, a nine inch disk); the other is a relatively small, very fast device (for example, a solid state device). The fast device is used as a cache.
- Logging container—This consists of three containers: a store, a backup copy, and a log. Updates are directed to both the store and the log. Reads are satisfied from the store. The store can be rebuilt from the log and the backup copy.

It is sometimes the case that all of the storage media represented by a container cannot be fully utilized by a single client, such as a file system or a database application. For this reason, containers can be sub-divided into *subcontainers*. Subcontainers offer the advantage of better media utilization, while retaining the properties of the container. For example, two relatively small file systems, both requiring high availability and reliability features can be created by sub-dividing a shadowed compound container.

From the above discussion on base, compound and sub-divided containers, it is clear that container abstractions can be realized in a general way. For example, several base containers can be combined to form a compound container, which can then be sub-divided to form several subcontainers. The lower portion of Figure 19 illustrates a few ways containers could be structured.

8.4.1.2 Device Drivers

A device driver is a collection of routines and data structures used by the OZIX system to process an I/O request. OZIX mass storage supports access to DSA (both DSA-1 and DSA-2) devices, as well as SCSI devices. Devices connected to an industry standard bus, such as Future Bus, may also be accessible in the future. The types of mass storage devices that are accessible are as follows:

- Magnetic disks
- Magnetic tapes
- Solid state devices
- Optical disks

- CDROM

OZIX users can directly access mass storage device drivers through either the *block device* interface or the *character device* interface. For block oriented devices like disks and tapes, a character device interface is quite unstructured and is therefore sometimes termed a *raw* interface.

DSA storage devices are accessible via the Computer Interconnect (CI) network, using the Storage Communication Architecture (SCA). SCA is divided into four distinct layers:

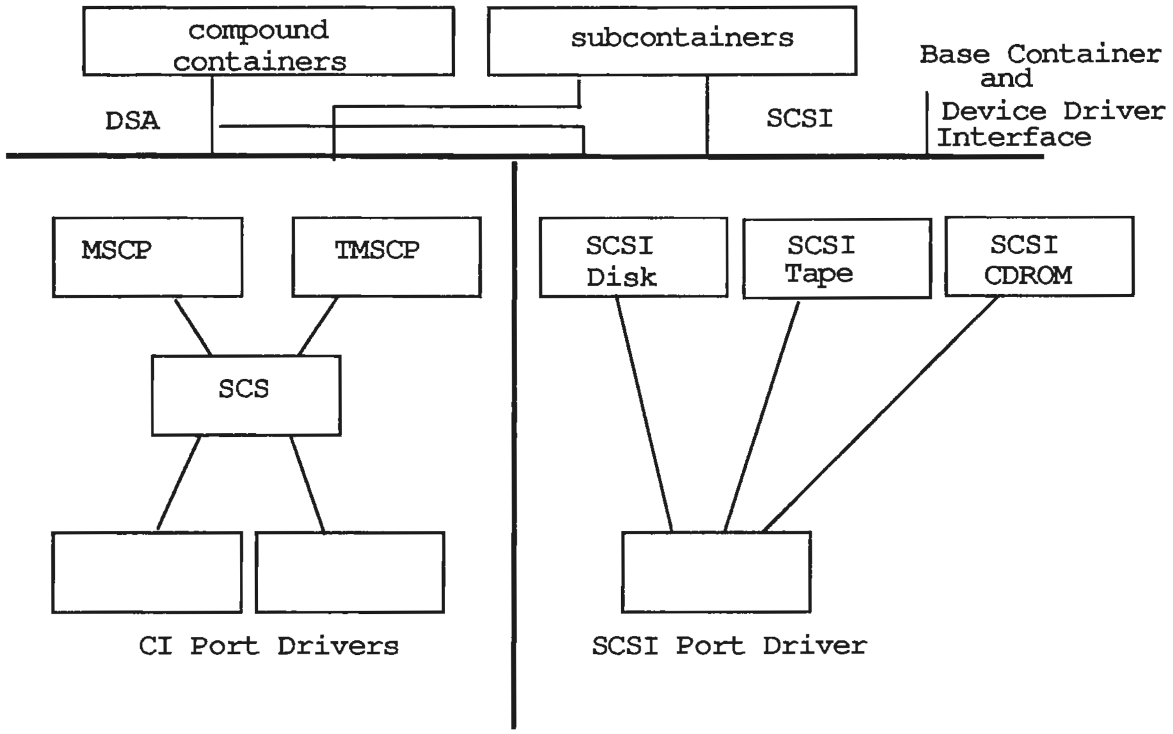
- Layer 0 is the Physical Interconnect layer
- Layer 1 is the Port/Port Driver (PPD) layer
- Layer 2 is the System Communication Services (SCS) layer
- Layer 3 is the System Application (SYSAP) layer

DSA device drivers are SYSAPs that use SCS to communicate with their counterparts residing in device controllers. Random access DSA devices are accessed through a mass storage control protocol (MSCP) class driver. Tape devices are accessed through a tape mass storage control protocol (TMSCP) class driver.

The system communication services (SCS) layer is responsible for managing connections between communicating SYSAPs. SCS performs some of the functions of the transport layer (layer 4), as well as some of the functions of the session layer (layer 5) of the OSI reference model.

The PPD layer is implemented as a combination of hardware and software. This layer performs some of the functions of the datalink layer (layer 2), the network layer (layer 3), and some of the functions of the transport layer (layer 4) of the OSI reference model.

Figure 22: Device Driver Components



8.4.2 Security

Mass storage components are required to handle data belonging to multiple users, as well as data belonging at multiple security levels. Thus, mass storage components must be defined as part of the trusted computing base (TCB). Being a part of the TCB imposes stringent requirements on the code. The mass storage components must satisfy the following requirements:

- Referring to objects—All objects must be accessed by way of the reference monitor
- Reusing objects— Any object that is reused must be (virtually or actually) zeroed
- Processing requests—All mass storage code must take the necessary precautions (for example, copy in a protected area) before processing a request

8.4.3 Management

OZIX I/O configuration manager (IOCM) provides the mechanisms to manage the hardware and software components of the OZIX mass storage system. The installed base of an OZIX system is expected to contain thousands of mass storage devices, all of which are to be monitored and controlled by the IOCM. The IOCM provides the following:

- A management interface that conforms to OSG Enterprise Management Framework
- System startup functions
- Adaptive management of I/O components on a running system

- System shutdown and restart functions

8.4.3.1 System Startup Functions

At the time an OZIX system is booted, IOCM does the following:

- Interacts with the software configuration manager to load and activate components of the I/O system
- Automatically configures physical devices
- Passes device and container information to the appropriate I/O modules to configure the mass storage subsystem

It is obvious from the above list that the IOCM has a significant role in configuring the OZIX mass storage system. IOCM startup deals with many software components, each having unique configuration requirements. Further, in order to meet the goals set for system startup time, IOCM must configure the mass storage system within a prescribed amount of time.

8.4.3.2 Running System

On a running system, the IOCM is required to provide the following functions, without disrupting regular services to the running applications:

- Load and unload components of the mass storage system
- Configure or auto-configure new physical devices
- Configure or re-configure containers

The IOCM can be queried for information regarding the states of the mass storage components.

CHAPTER 9

TERMINAL SUBSYSTEMS

9.1 Introduction

The OZIX terminal subsystems are designed to meet the requirements of transaction processing (TP) systems, which process hundreds to thousands of transactions per second. Systems that process such large volumes of transactions require tens of thousands of terminal sessions.

As OZIX is an internationalized product, the OZIX terminal strategy provides a method for supporting different character sets and languages. Specifically, the OZIX terminal subsystem supports multi-byte characters sets, which are prevalent in Asia.

Terminals are mostly used in *Canonical mode*. In this mode, input characters are echoed by the system as they are typed and are collected until a carriage return is received. In *Non-Canonical mode*, the terminal subsystem does not process special characters, and passes all characters to the application program reading from the terminal.

The following requirements have been identified for the terminal subsystem:

- Conformance to OSF and POSIX standards for interactions with terminals
- Support multiple character sets
- Support diverse terminal connection mechanism
- Conform to OZIX system management
- Conform to OZIX security model

9.1.1 Goals

The terminal subsystem has the following goals:

- Conform to OSF and POSIX standards for terminals.

  This goal is realized by supporting all of the terminal attributes defined by POSIX, including those for process control and timed input.
- Internationalization support.

  The terminal subsystem will support input and output of multi-octet characters which are needed to support many of the world's languages. In addition, individual character-set/language modules may handle the needs of particular languages. For instance, a method of converting phonetic input into Japanese Kanji characters will be supplied as a "character-set specific" module that plugs into the OZIX terminal driver
- Support for different connection methods.

Terminals may be connected to an OZIX system in a number of different ways. It should be possible to support a new connection method without having to rewrite the terminal subsystem. The primary method of connecting terminals to an OZIX system will be through the use of terminal servers which run network protocols such as LAT or TELNET.

- Scalability.

  The terminal subsystem is designed to support tens of thousands of terminal sessions. To support this, there will be a method of creating terminal connections "on the fly" without the need for a preconfigured terminal database.

- Provide for a robust I/O configuration management.

  This goal is realized by conforming to the EMA specifications. The terminal subsystem has a management interface, as well as descriptions for all terminal subsystem objects and object attributes.

- Provide support for the line disciplines specified in POSIX

The following are non-goals for version 1.0 OZIX terminal subsystem:

- Direct support for FORMS
- Provide support for real time I/O
- Support line disciplines other than that specified in POSIX
- Support modems

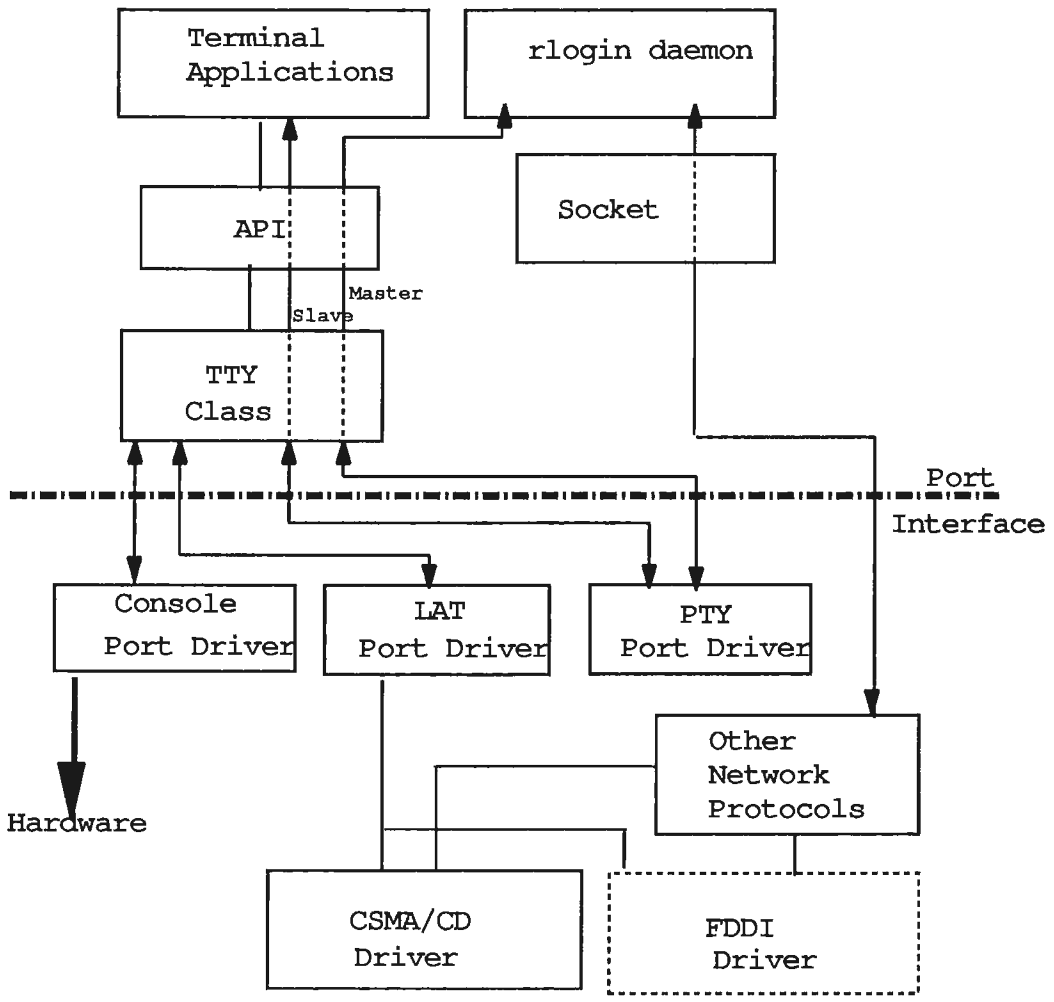
9.2 Terminal Components

OZIX terminal subsystem is divided into two major modules:

- Terminal Class Driver—This driver is responsible for providing user sessions as well as functions for data presentation (thus representing the session layer and the presentation layer of the OSI reference model). In OZIX V1, the terminal subsystem provides only the TTY class driver, which follows presentation rules specified by POSIX standard 1003.1.
- Terminal Port Driver—This driver is responsible for moving uninterpreted data between a hardware interface or network software module and the terminal class driver.

Figure 23 shows various components of the terminal subsystem along with the network drivers.

Figure 23: Terminal Subsystem and Network Drivers



Terminals may be connected to an OZIX system in a number of different ways. Connection methods include:

- Console terminals
- LAT terminals
- Network terminals (TELNET, rlogin, CTERM)

A Console terminal port driver interacts with the hardware console subsystem to make console terminal a path through which a system manager, maintenance engineer, or system user can communicate with the OZIX system.

OZIX provides local area networks to facilitate high speed communications within a relatively short distance (for example, a campus). OZIX local area network uses a physical interconnect, which is a single, shared network channel. This physical interconnect is used by network interface drivers such as the Ethernet driver and the FDDI driver.

The LAT provides functions similar to the OSI reference model network layer, transport layer and session layer. The LAT uses the network interface drivers to provide LAT terminal ports.

OZIX provides a pseudo terminal (PTY) port driver to simulate the actions of terminals in software. PTY provides the initial support for TELNET, rlogin and CTERM.

A *pseudo terminal* consists of a pair of character devices: a *master* device and a *slave* device. The slave device appears to OZIX as a normal interactive terminal. The master device is used by an application program to simulate a human typing on an interactive terminal. The Figure 23 shows a remote login (rlogin) daemon interacting with the various terminal subsystem and network subsystem components. When a request for a remote login session is received, the rlogin daemon allocates a pseudo terminal and manipulates the file descriptors so that the slave half of the pseudo terminal becomes the stdin, stdout, and stderr for a login process. The PTY driver funnels writes on the slave device as input to the master half of the pseudo terminal. The input to the master is eventually carried to the remote host by way of the sockets and the underlying network. Similarly, anything written on the master half of the pseudo terminal is given to the slave device as input.

The network interface drivers provide datalink services to higher level software, such as DECnet Phase V and TCP/IP components.

9.3 Security

OZIX terminal subsystem is designed so that most of the code is not required to handle data from multiple security levels. Thus, most of the terminal subsystem consists of untrusted code. However, any code that deals with data from multiple security levels is part of the trusted computing base. In addition, the OZIX terminal subsystem will provide a way for a user to connect to the Trusted Computing Base in order to determine or change his security classification.

9.4 Terminal Component Management

The I/O configuration manager is a software application responsible managing OZIX terminal subsystem components. These management functions include:

- Loading, unloading, or replacing port driver modules
- Loading, unloading, or replacing character-set support modules
- Defining special key maps for character-set support

The terminal subsystem co-operates with the IOCM to meet the system goals of 7X24 operation and system recovery time.

CHAPTER 10

NETWORK IMPLEMENTATION ARCHITECTURE

10.1 Introduction

Networks and distributed systems technology is at the heart of Digital's open systems products. As an integral part of OSG, the OZIX group is pioneering a distributed application platform for the 1990s. We expect that application developers, both internal and third party, will use this platform to build their applications.

The underlying design of the network will result in a powerful set of products upon which customers can rely for high performance and reliability. The network design is enhanced by features integrated into the operating system, such as support for transaction processing, journaling and recoverability primitives in the file system, remote procedure call, and authentication services.

An important part of the OZIX architecture is reflected in our dependence on and adherence to OSF, POSIX, ISO, Internet, and other standards. In some cases we will provide implementations that support multiple solutions, for instance, in the areas of authentication and name service. We look to NaC to assist us in providing industry-leading solutions to these challenging integration problems.

This chapter presents the goals and strategy for the OZIX network architecture and a description of the major components. These components form only part of the technology upon which distributed applications are built. See the *Base System Architecture* and *RPC Architecture* documents for descriptions of additional OZIX features that are part of OZIX's distributed applications environment.

10.2 Goals

The goals of the OSG network strategy are to:

- Exploit distributed systems and networks for Digital's advantage in the open systems market
- Support one family of open systems products in a network
- Provide a distributed computing environment using *internet* and OSI standards
- Provide DECnet/OSI distributed computing integration
- Lead the migration of open systems to OSI standards
- Propose and drive open systems standards for distributed systems
- Provide connectivity with VMS systems without being VMS-dependent

10.3 Strategy

OSG products, which include workstations and servers, share a common network and distributed systems architecture. In accordance with this architecture, OZIX will implement the network components and distributed services for the server environment.

- Digital's open systems products will implement this distributed systems architecture using TCP/IP-based extended local area network (LAN) subnets. Wide area network access will be provided through a DECnet/OSI backbone using the Internet Portal product.
- OZIX systems will provide *Phase V DECnet / OSI* end-node support, making them accessible to Phase V-compliant systems anywhere on the extended Phase V network.
- Interoperation between VMS and internet protocols on the subnet will be available through the *VMS / ULTRIX Connection*, a VMS layered product. Communication with VMS systems not on the subnet will be available through a DECnet/OSI backbone using the Internet Portal product.
- The *network application programming interface* (API) presented to applications will be common across internet transport protocols, ISO transport, and DECnet/OSI Session Control for OZIX and ULTRIX. It is not a goal of the API to hide semantic differences among the transport protocols.
- OZIX systems will provide a network management interface that is integrated with system management. Network management of DECnet/OSI and TCP/IP will be integrated at the user interface. The network management programming interface is common on OZIX and ULTRIX.
- Digital's open systems will adhere to OSF standards. Where deficiencies are noted, Digital will propose its distributed systems architecture components. Digital will lead the market by integrating these services.
- Workstation services include:
  - Name services using *Distributed Name Service* (*DNS*) and *Berkeley Internet Name Domain Service* (*BIND*)
  - Time services using *Digital Time Synchronization Services* (*DTSS*) and *Network Time Protocol* (*NTP*)
  - File services using *Network File System* (*NFS*)
  - Wide area network access through DECnet/OSI routers and x.25 gateways
  - Authentication services using *Kerberos* and *Digital Authentication Security Service* (*DASS*)
  - Remote procedure call services using *DEC RPC*
- Digital will accelerate the movement of the market toward OSI by providing an OSF reference implementation that is compliant with OSI standards.
- Digital will add value in the distributed systems area by providing solutions in naming, network management, RPC, security, authentication, common application subsystem interfaces, and time services.
- Digital's open systems will provide x.25 access and SNA interoperation through low-cost gateways, consistent with the corporate strategy. If direct connection becomes important in certain markets, third-party solutions will provide this capability.
- PC integration for Digital's open systems will be provided by DECnet/OSI and third-party solutions.

10.4 Performance

The OZIX network will be co-designed with the operating system in order to fully utilize the advanced capabilities provided by OZIX. The network will provide state-of-the-art performance by exploiting the inherent parallelism of symmetric multi-processing (SMP) through the use of the OZIX thread architecture.

The effective use of any thread architecture requires that it be designed into the network implementation, since it affects the structure of the internal network database, influences the choice of an appropriate database locking granularity, and demands the studious avoidance of unnecessary thread context switching.

As a result of these considerations, the OZIX network will enjoy a significant performance advantage when compared to implementations that do not employ threads as a fundamental design element.

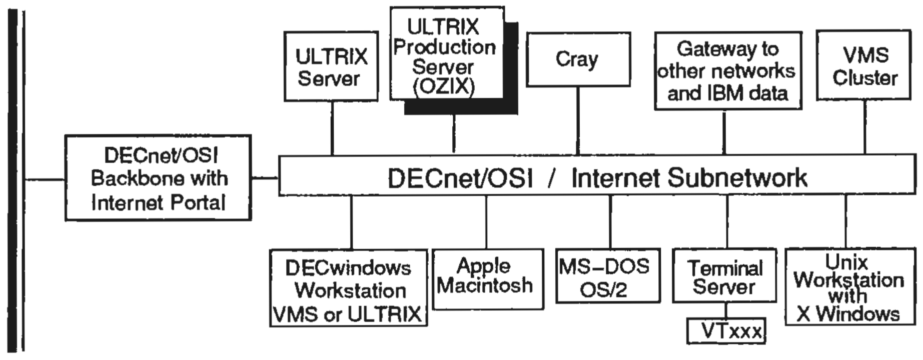
In order to ensure the performance advantage gained by the intrinsic parallelism of the OZIX network, careful attention is given to the fundamental issues applicable to any good network design. These include, for example, the use of buffer management mechanisms that avoid data copying wherever possible, as well as the extensive use of direct procedure calls between network layers.

10.5 Architectural Overview

The OZIX networking environment is designed to support heterogeneous systems (workstations and servers) participating in both local and wide area networks. The product is a combination of DECnet/OSI Phase V and internet components. Wide area network access is provided by a DECnet/OSI backbone gateway and an internet portal. TCP/IP is the protocol used for ULTRIX interoperation on the local area network; DECnet/OSI provides interoperability with VMS systems and other vendors' OSI end nodes (see Figure 24). The ultimate goal is to encourage the industry to move toward OSI, thus a single network solution. We look to NaC to provide leadership in influencing this migration to OSI.

OZIX strives to provide Internet functionality matching or exceeding any other competitor; it will meet all relevant IP standards defined at least 6 months prior to field test.

Figure 24: Network Architecture Overview


ULTRIX Server
ULTRIX Production Server (OZIX)
Cray
Gateway to other networks and IBM data
VMS Cluster
DECnet/OSI Backbone with Internet Portal
DECnet/OSI / Internet Subnetwork
DECwindows Workstation VMS or ULTRIX
Apple Macintosh
MS-DOS OS/2
Terminal Server
VTxxx
Unix Workstation with X Windows

10.6 Internet Components

Six protocols form the basis for an internet network node (see Figure 25). Internet networks do not provide distinct protocols at ISO's session and presentation layers.

At the transport layer:

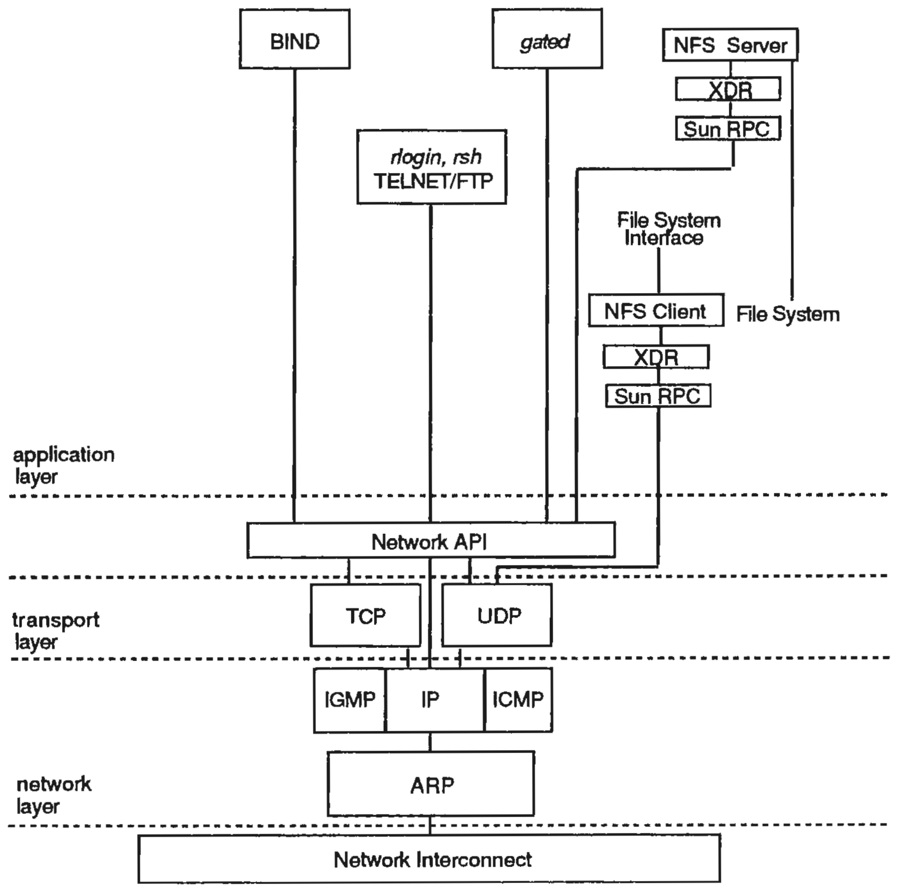
- *Transmission Control Protocol* (*TCP*)
- *User Datagram Protocol* (*UDP*)

At the network layer:

- *Internet Protocol* (*IP*)
- *Internet Control Message Protocol* (*ICMP*)
- *Internet Group Management Protocol* (*IGMP*)
- *Address Resolution Protocol* (*ARP*) (includes *Reverse Address Resolution Protocol* (*RARP*) and *Proxy ARP*)

The following sections detail each layer and its associated protocols.

Figure 25: Internet Network Components



10.6.1 Application Layer Code

Application code accesses OZIX network services through the network API which includes both Berkeley *sockets* and the *X/OPEN Transport Interface* (*XTI*). Portable application code using either of these interfaces can be moved to OZIX.

The Berkeley Internet Name Domain (BIND) Service, an implementation of the Internet Domain Name Service, is ported from BSD4.3. BIND is a network service that allows client systems to obtain translations of host names into internet addresses.

It is expected that the Open Systems Components and Resources (OSCR) group will maintain the set of common application codes for both OZIX and ULTRIX/OSF.

10.6.2 Transmission Control Protocol (TCP) Module

At the transport layer, TCP provides a reliable, connection-oriented service that guarantees that messages are delivered to their destination in the order they were originally sent. It uses the network service of the Internet Protocol (IP) to communicate between peer TCP protocol layers.

10.6.3 User Datagram Protocol (UDP) Module

At the transport layer, UDP provides an unreliable, connectionless service commonly referred to as a *datagram* service. Applications using this service can send datagrams to any other UDP user, either local or remote. Because it is an unreliable service, datagrams may be lost, delivered out of order, or may be duplicated.

An outgoing datagram can be sent to a single UDP session on a distinct system, or it can be broadcast to many systems. The semantics of broadcast transmission, reception, and propagation are implemented in the IP network layer.

UDP datagrams can also be sent to and received from IP multicast groups. A multicast group is a special class of Internet addressing used by cooperating hosts to provide broadcast-like transmission and reception without the cost normally associated with broadcasting. The members of group are the only hosts that incur the cost of processing the multicast datagram.

10.6.4 Internet Protocol (IP), Internet Control Message Protocol (ICMP), and Internet Group Management Protocol (IGMP) Modules

The Internet Protocol (IP) provides unreliable datagram services at the network layer. IP fragments and reassembles datagrams that are larger than those supported by lower network layers.

IP moves datagrams between nodes. The outgoing network interface and the IP address of the next-hop are found by looking in IP's routing database. This database is updated by a combination of ICMP redirect messages and an application subsystem program (see the discussion of *gated* in Section 10.6.5). If a node resides on two networks, IP can also forward datagrams not destined for one of the local internet addresses. This feature can be enabled to create a routing node.

TCP and UDP are layered on top of IP. The destination transport protocol (either TCP or UDP) is selected by a protocol ID in the IP datagram. IP is subsequently layered on any subnet-dependent device that can support IP's concepts of addressing and datablocking. Device-dependent convergence layers provide the mapping of IP addresses to the subnet-dependent device's addressing scheme. OZIX supports the Address Resolution Protocol (ARP) (discussed in Section 10.6.6), which is responsible for mapping internet (logical) addresses to Ethernet and IEEE 802.3 (physical) addresses.

IP is not designed to provide reliable delivery. The Internet Control Message Protocol (ICMP) is used to provide feedback in the internet communications environment. Since ICMP is based on IP for its communications, an ICMP datagram has all the characteristics of a normal IP datagram, for example, it may not be delivered. ICMP is an integral part of IP and must be implemented by every IP module.

The Internet Group Management Protocol (IGMP) is used by IP hosts to communicate their multicast group memberships to any immediately neighboring multicast routers. Currently, multicast routing is experimental in the Internet, but the protocol is required for Level 2 IP multicast support. Due to their experimental classification, multicast routers are currently not supported on OZIX.

10.6.5 Routing Database Maintenance

The routing database management daemon, *gated*, is from Cornell University. It accesses IP through the network API and receives the External Gateway Protocol (EGP), Routing Information Protocol (RIP), Defense Communication Network's HELLO routing protocol, and Internet Control Message Protocol (ICMP) messages that affect the routing database. It interprets these messages and communicates directly with IP to change the internal copy of the routing database.

A new internal routing protocol, Open Shortest Path First (OSPF), is being introduced in the Internet community. OZIX will track its use and, if appropriate, provide an implementation of OSPF at FRS.

10.6.6 Address Resolution Protocol (ARP) and Proxy ARP Modules

At the network layer, Address Resolution Protocol (ARP) converts 32-bit internet addresses to 48-bit Ethernet and IEEE 802.3 addresses. Reverse ARP translates Ethernet and IEEE 802.3 addresses to internet addresses.

Proxy ARP allows a gateway to answer ARP requests on behalf of a destination node not on the same subnet.

10.7 DECnet/OSI Components

The components that make up DECnet/OSI on OZIX are presented according to their layer in the network model (see Figure 26).

At the application layer:

- *File Transfer and Access Management (FTAM)*
- *Virtual Terminal Protocol (VTP)*
- *DECnet/OSI Copy Program (DCP)*
- *dlogin*
- *Distributed Name Service (DNS) Clerk*
- *Digital Time Synchronization Services (DTSS) Client*
- x.400

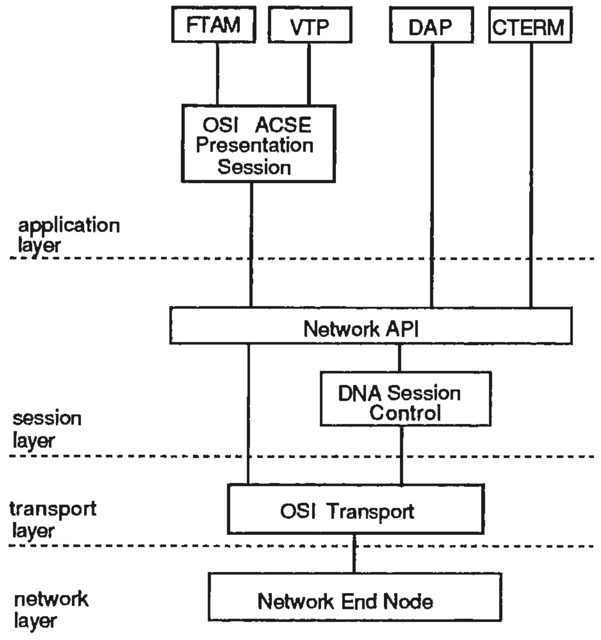
At the session layer:

- DNA Session
- OSI Session

At the transport layer:

- OSI Transport Protocol (TP4)

Figure 26: DECnet/OSI Network Components



10.7.1 Application Level Components

The application level components that OZIX provides are a mixture of DECnet-specified and OSI-specified components. DECnet-specified components will be ported from ULTRIX. OSI-specified components may not exist in a portable form and may have to be written or purchased from a third party.

10.7.1.1 File Transfer and Access Management (FTAM)

FTAM is the OSI method for file transfer and access over the network. The ULTRIX version of FTAM will be ported to OZIX.

10.7.1.2 Virtual Terminal Protocol (VTP)

VTP is the OSI virtual terminal protocol used to request outbound remote terminal sessions. VTP will be ported from ULTRIX.

10.7.1.3 DECnet/OSI Copy Program (DCP) and the File Access Listener (FAL) Daemon

DCP is the client portion of the DECnet/ULTRIX file transfer utility. The FAL daemon is the server portion, and responds to DCP requests. This facility is used to interchange files between OZIX systems or systems running DECnet/OSI Phase V.

10.7.1.4 dlogin and the dlogind daemon

dlogin is the DECnet virtual terminal program from DECnet/ULTRIX. It is used to request outbound remote terminal sessions. The *dlogind* daemon is used to accept inbound terminal sessions from remote DECnet/OSI systems. Together they implement the DECnet/OSI CTERM virtual terminal protocol. *dlogin* and *dlogind* will be ported from ULTRIX.

10.7.1.5 Distributed Name Service (DNS) Clerk

The DNS Clerk is the local interface to the Distributed Name Service. It is required for DECnet/OSI Phase V. It will be provided for OZIX by Distributed Process Engineering (DPE).

10.7.1.6 Digital Time Synchronization Service (DTSS) Client

The DTSS Client is the local interface to the DTSS server. It is required by the Distributed Name Server, which is itself required for DECnet/OSI Phase V. It will be provided for OZIX by DPE.

10.7.2 Session Layer Components

10.7.2.1 DNA Session

DNA Session is the DECnet/OSI Phase V session control layer. It allow communication capability to those applications that require the full DNA protocol stack.

10.7.2.2 OSI Association Control Service Element (ACSE), Presentation, and Session

The upper layers of OSI (ACSE, presentation, and session) allow communication capability between applications that use the full OSI protocol stack. Together, they constitute the preferred interface between DECnet/OSI Phase V systems and other OSI-compliant systems.

10.7.3 Transport Layer Component

DECnet/OSI supports ISO's *Transport Class 4* (*TP4*), which is the most robust of the ISO-specified transports. TP4 provides the capability to multiplex many concurrent network sessions over an unreliable physical link.

The ISO TP0 and TP2 transport classes, which support non-multiplexed (TP0) and multiplexed (TP2) communications on reliable, error-free physical links, will be considered for future releases of OZIX.

10.7.4 DNA Network Layer

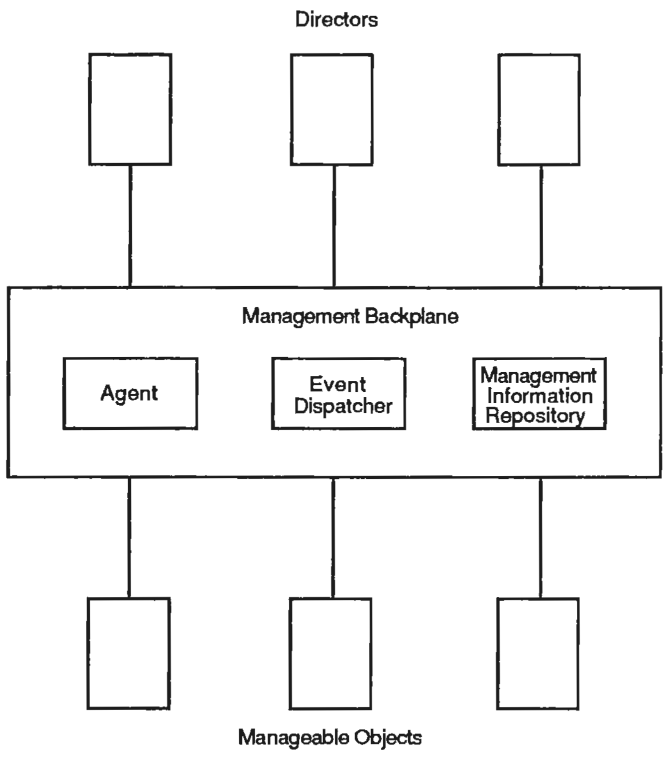
The network layer (end node routing) on OZIX conforms to the DECnet/OSI Phase V network layer architecture and supports DECnet/OSI Phase V-style addresses. The OZIX implementation of the network layer provides end node support only. Support for multiple physical links on Ethernet (multi-link) will be provided to improve throughput and availability.

10.8 Management Components

In OZIX, system and network management are integrated to present a seamless, coherent management picture. System and network components are managed together using the same *management backplane*. Local and remote objects can be managed using this backplane.

The backplane provides dynamically installable protocol towers for communication that conforms to DNA and Internet specifications. Currently the protocols supported are *Common Management Information Protocol (CMIP)*, *CMOT* (CMIP over TCP/IP), and *Simple Network Management Protocol (SNMP)*.

Figure 27: Network Management Components



10.8.1 The Backplane

In Figure 27, directors manage the system and *network objects* via the backplane services. The backplane provides a common API to the directors for directives, and a common API to the *manageable objects*.

10.8.2 The Director

The *director* is an application-level program that generates management requests. The interface is graphical in nature, although it derives from a textual-based system, *Network Control Language (NCL)*. NCL will be extended and renamed *MCL (Management Control Language)* to incorporate both system and network management directives. Internet, DECnet, and system manageable objects will all be manageable by this director.

10.8.3 The Agent

The *agent* is the intermediate recipient of all management directives. The agent locates and passes the request to the appropriate manageable object.

10.8.4 The Event Dispatcher

The *event dispatcher* (*EVD*) is a system component that accepts internal state change notification from any manageable object in the system. It dispatches the event to local and remote systems for further processing. The event dispatcher conforms to the DNA Phase V EVD functional specification, although its functionality is extended to encompass both system and network event postings.

10.9 Network File System

The *Network File System* (*NFS*) is a collection of components that provide transparent access to files located on remote systems. These components are based on the Internet protocol suite, as well as de facto standards that originated from Sun Microsystems. This implementation uses V2 of the NFS file protocol.

NFS is based on the V4 release of the reference code from Sun. Some portions will be rewritten to accommodate performance, B2 security, multithreading, and internationalization.

10.9.1 Sun Remote Procedure Call

The NFS components communicate with one another using the *Sun Remote Procedure Call* (*RPC*) mechanism. This allows programs to model remote communication as simple procedure calls. Sun RPC uses the *External Data Representation* (*XDR*) library to convert OZIX data forms into a canonical form so that any implementation can properly interpret the data.

Sun RPC and XDR are provided for use by NFS and are not supported for customer use.

10.9.2 NFS Client Subsystem

The NFS client gives OZIX applications access to files that are exported by remote systems. The application interface to the NFS client is identical to that of the native OZIX file system. The NFS client converts OZIX file I/O requests for remote files into network requests. A remote NFS server performs the actual file accesses on behalf of the requesting application.

The NFS Client communicates with the User Datagram Protocol (UDP) subsystem using the network transport interface instead of using the network Application Programming Interface (API).

10.9.3 NFS Server Subsystem

The NFS file server subsystem accepts client requests from remote nodes and executes them as local file system requests. The server will implement the duplicate request filtering enhancements first introduced with ULTRIX.

Two other subsystems assist the file server. The port mapping server allows callers to properly locate the server; the filesystem mounting server distributes initial access to exported filesystems.

10.9.4 NFS Locking Server Subsystem

The NFS locking subsystem allows clients on remote nodes to lock files on the system where the locking server is running. The locking server works with the status monitor subsystem, which is responsible for connection management issues.

From a client's perspective, if a lock is requested on a local file, OZIX performs the requested action directly. If the lock is on an NFS file being served by a remote system, a locking request is sent to the locking server on the remote system.

10.9.5 NFS Management

NFS management is integrated with the OZIX system management framework. Generic management of each of the NFS components consists of starting the component, stopping the component, getting attributes and setting attributes.

Management of the NFS client is accomplished through management of "filesystem objects." A filesystem is a directory subtree made available on a system through NFS. *Mounting* is the creation of a new NFS filesystem, and *unmounting* is the removal of an NFS filesystem. An NFS filesystem is managed by getting and setting its attributes.

Management of the NFS server is accomplished through management of "exported filesystem objects." An exported filesystem is an entire local filesystem, directories, or a single file which is made available for access by remote clients. An exported filesystem is managed by getting and setting its attributes.

10.9.6 Techniques used for NFS Performance

The following techniques are used to increase NFS performance.

- Sun RPC uses the network transport interface to UDP which minimizes copying of data.
- The NFS client uses the filesystem buffering mechanisms for read ahead and write behind.
- The NFS server uses the filesystem caching mechanisms to avoid unnecessary disk accesses.
- The NFS server uses multiple threads to increase concurrency.

10.9.7 Highly Available File Service

The OZIX availability model uses a partitioned approach wherein a subset of the resources provide service at any given time. A secondary server is designated to recover state and resume service. The NFS components are stateless in that there is no required state to be recovered upon restart after a failure. The NFS components work with other highly available OZIX components such as the network transport, file system, and mass storage subsystems.

10.10 DEC Remote Procedure Call

A remote procedure call capability will be provided by *DEC RPC* V1.0. The ability to write applications using remote procedure calls facilitates development of distributed applications by allowing programs on various nodes across a local area network to register as providers of services, or *servers*. Consumers of these services, or *clients*, can then make requests of any server that has been registered as a provider of the desired service.

DEC RPC V1.0, which is based on Apollo's *Network Computer System* (*NCS*) version 1.5, is the first product offering defined by the joint DEC/Apollo RPC Architecture and will be bundled with ULTRIX V4.0 as well as with V1.4 of the VMS/ULTRIX Connection (UCX).

DEC and Apollo are jointly submitting RPC V2.0, an enhanced version of DEC RPC V1.0, to OSF in response to the Distributed Application Environment (DAE) Request for Technology (RFT). It is expected that RPC V2.0 will be the RPC chosen by OSF, although because of time constraints, OSF will likely include NCS V1.5 in OSF/1.

By providing DEC RPC V1.0, OZIX will be able to interoperate with ULTRIX and Apollo systems. If OSF accepts the joint DEC/Apollo RPC V2.0 submission, as it is fully expected to do, OZIX's level of interoperability will be greatly increased.

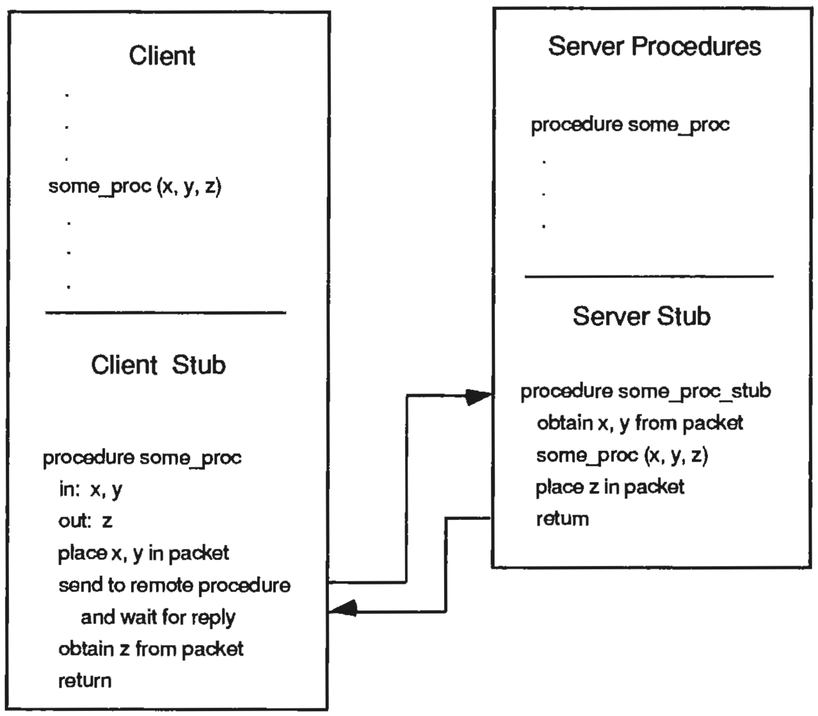
DEC RPC V1.0 uses the Berkeley socket interface to the UDP transport layer. It includes three major components: the *Network Interface Definition Language (NIDL)* Compiler, the RPC runtime library, and the *Location Broker*.

10.10.1 Network Interface Definition Language (NIDL) Compiler

The NIDL Compiler is the stub generator for DEC RPC. An NIDL interface definition rigorously describes the interface to a remote procedure: it defines the functions, procedures, and data types of the parameters that make up a remote interface.

Given an interface definition written in the Network Interface Definition Language, the NIDL Compiler produces C header files, and client and server stubs with the appropriate RPC runtime library calls. Figure 28 illustrates the role of client and server stub code in a remote procedure call.

Figure 28: Client and Server Stub Interaction



10.10.2 The RPC Runtime Library

The RPC runtime library actually implements the RPC functions. It is composed of routines that affect bindings, register and unregister remote interfaces, marshall parameters, provide server information, and access the network. Many of these routines are never directly called by clients or servers, but are instead called from the stubs that are generated by the NIDL Compiler.

10.10.3 The Location Broker

A server database is maintained by the DEC RPC Location Broker, allowing it to associate incoming client requests with an appropriate server. This mechanism requires that servers register information such as socket addresses and exported procedure definitions with the Location Broker.

It is expected that in future releases, DEC RPC will move towards using the Distributed Name Service (DNS), but in the initial release, it will use the Location Broker.

10.11 Network Interconnects

Initially OZIX will support only Ethernet and IEEE 802.3 as communications links. This is consistent with our strategy to provide wide area network access through Digital's OSI backbone gateways and IP portal products. OZIX's DECnet/OSI product will support only standard IEEE 802.3 and accommodate multiple controllers. Internet will use both IEEE 802.3 and Ethernet, but considers them logically separate network interconnects. Internet will also support multiple controllers.

As fiber distributed data interface (FDDI) products are made available to the hardware platforms supported by OZIX, FDDI will be incorporated as a network interconnect for both Internet and DECnet/OSI.

Direct access to the network interconnects is provided via two interfaces: the Data Link Interface (DLI) and the Packet Filter.

The DLI is presented as a distinct address family in the BSD socket portion of the network API. DLI endpoints are not available through the XTI portion. The appropriate access controls are used to prevent unauthorized use of the devices represented by the DLI. The functionality provided by an OZIX DLI endpoint is identical to that of ULTRIX DLI.

The Packet Filter is similar in function to the DLI, but provides a richer filtering mechanism[1]. The Packet Filter allows users to specify multiple, non-contiguous fields in a packet as the selection criteria. This is compared to the DLI wherein only one field is used for demultiplexing. The Packet Filter was designed to allow prototyping of network code in user mode while providing a relatively low overhead interface to the network.

10.12 IBM Interconnect Components

The components of the OZIX IBM Interconnect will make use of the planned DECnet/OSI Phase V SNA gateway products. The components fall into two categories; application components that can be used generically over OZIX or ULTRIX, taking advantage of the common API Libraries; and those that must be ported from ULTRIX- or VMS-based versions.

10.12.1 Generic IBM Interconnect Components

Applications interacting with the SNA gateway do so via DECnet/OSI. Hence, the portable applications and libraries already planned by the SNA Interconnect Group will be able to use the XTI and/or socket interfaces to DECnet/OSI/OZIX without modification. These products include:

- DTF File Transfer - Allows file transfer between hosts on connected DECnet/OSI and SNA Networks.
- DW 3270 Terminal Emulation - Provides emulation of IBM's 3270-style terminals on DEC terminals and DECwindows.

[1] see "The Packet Filter: An Efficient Mechanism for User-level Network Code", Jeffery C. Mogul, Richard R. Rashid, and Michael H. Accetta, Proceedings of the 11th Symposium on Operating Systems Principals, Austin, Texas, November 1987.

- 3270 Data Stream - Provides a programming interface for our users to interact with SNA host-based applications designed to be accessed via 3270 style terminals.
- SNA Application Programming Interface - Provides a programming interface for our users to interact with SNA using LU0 protocol.
- APPC/LU6.2 - Provides a programming interface for our users to interact with SNA using LU6.2 protocol.

10.12.2 Ported IBM Interconnect Components

Most components that may require some conversion effort include managing the SNA gateway products.

SNA gateway management consists of downline-loading an image for a given SNA gateway hardware product, setting, reading, and changing its manageable objects once it is running, and monitoring its operation.

10.12.2.1 Booting the SNA Gateway

An SNA gateway hardware product will be booted via the *Maintenance Operations Protocol* (*MOP*), already planned as a part of the DECnet/OSI product for OZIX. No additional work will be required.

10.12.2.2 SNA Gateway Management

The manageable entities within an SNA gateway product will have to be understood by our network management utilities. Access to these manageable entities will be via the CMIP protocol, already planned for OZIX as part of the DECnet/OSI network management implementation. Definitions of the manageable entities will have to be loaded, but the definitions are expected to be independent of the operating system.

10.12.2.3 SNA Gateway Monitoring

Two monitoring applications are required and will have to be ported from other operating systems. These are the Common Trace Facility (CTF), which includes the capability to trace SNA gateway line activity, and SNAP, a gateway monitoring tool.

10.12.3 Access to OZIX via SNA

Products planned by Digital to facilitate access to data and applications from the SNA side of the gateway use existing protocols already planned for inclusion in OZIX. These products and their associated protocols are:

- Medusa (LAT) - Allows 3270 style terminals access to either IBM resources, or LAT access to DEC resources via Ethernet.

10.13 Application Programming Interface (API)

Two application programming interfaces (APIs) are provided to all OZIX-supported transport layers. The first form was developed at Berkeley for the BSD 4.x operating system and is called *sockets*. The second form was developed at AT&T for the System Vr3 operating system and is called the Transport Layer Interface (TLI). The TLI has been adopted and slightly changed by X/OPEN; it is now called X/OPEN Transport Interface (XTI).

OZIX provides both of these interfaces as part of the OSF API. The socket/XTI API is designed to be a very thin veneer over the network transport interface so as to not impact performance of the applications. All OZIX transport layers implement the network transport interface, which is the union of socket and XTI functions.

While there is a common API across DECnet/OSI and internet, the specification of the transport to be used is part of the call to the API. The issue of making this transparent to the caller is a potential area for OSG to contribute added value.

10.14 Network Security

10.14.1 Philosophy

The goal for OZIX to be both secure and interoperable presents an interesting challenge since, at this time, there are no widely accepted intravendor protocols that provide a reasonable level of security. Therefore, OZIX will take a phased approach to network security. In the first release, only a single network operating at a single security label will be provided.

Over a period of several releases, and as some of the expected security protocols become standardized, OZIX will provide more extensive support for multilevel network capabilities.

10.14.2 The Trusted Computing Base (TCB) Boundary

The networking software will not be trusted in the first release of OZIX. However, subsequent releases will require that some or all of the network software will be a part of the TCB. The network layer and at least some of the transport layer will be required to handle data from multiple layers simultaneously. Any cryptographic routines will have to be trusted, as these routines will be moving data between two security labels. Some authentication routines will also have to be trusted.

10.14.3 Performance Options and Tradeoffs

Many options are available to the networking software to allow tradeoffs between security, performance, and functionality. Some of these tradeoffs are listed below.

- Including more code in the TCB will reduce the number of trusted/untrusted transitions, resulting in increased performance. Not all network code needs to be in the TCB, but if any untrusted code resides in a performance-critical path between two trusted portions of the network, then it may be included in the TCB to improve efficiency.
- The specifications for the first release of OZIX networking are extremely restrictive. This is required to pass the government certification requirements. However, not all customers require this level of security. As an option, the security label at which the network runs could become a range of security labels, with the understanding that any data going over the network is likely to lose its exact label, and have a new one assigned by the receiving system.
- Additionally, authentication requirements could be relaxed for some customers, so that the username/password or soft proxy information sent by existing protocols could be trusted, giving access to a wider range of target user accounts.

10.15 Services

10.15.1 Authentication Services

OZIX will implement kerberos as its authentication mechanism. This will be achieved by modifying NFS to authenticate clients and servers using kerberos.

Kerberos currently resides in public domain and is being implemented by both Apollo and IBM. It has also been presented to OSF. While kerberos will serve as the initial authentication mechanism for OZIX, NaC's *Distributed System Security Architecture* (*DSSA*) will need to be integrated into OZIX. How this will be accomplished has not yet been determined.

10.15.2 Name/Directory Services

Several name services have been accepted, proposed, or required to support the protocols and services which are part of the OZIX strategy. These include the Internet Domain Name Service, Distributed Name Service (DNS), x.500, and the location broker for NCS.

OZIX and ULTRIX will support the integrated DNS/*hesiod*/BIND name server being developed by Distributed Process Engineering(DPE). This name server supports DNS semantics (as required by DECnet) and a BIND interface (as required by Internet), thereby supporting both networks. *Hesiod*, developed by project Athena at MIT, is an enhanced user interface to BIND. These, in combination with the back end update characteristics of DNS, will provide us with a superior name server. This name server will be presented to OSF by NaC.

10.15.3 Mail Services

Mail Services will be provided by Open Software Components and Resources (OSCR), the common software group within OSG. They are developing an x.400-based product that uses DNS and x.500 directory services. In addition, it will support *Simple Mail Transport Protocol* (*SMTP*) and *Message Router* (*MR*) interfaces. This product will be common for both OZIX and ULTRIX.

10.15.4 Boot Services

Ozix will support Digital's Maintenance Operations Protocol (MOP). The internet booting protocol, *BOOTP*, will be supported, although the method of support is a marketing issue to be resolved at the appropriate time. BOOTP is a proposed standard documented in RFCs 951 and 1084. BOOTP thus far has not been adopted by other major workstation competitors such as Apollo.

An alternative strategy being investigated would be to support the extensions to RARP as developed by SUN.[2] These extensions are commonly known as *BOOTPARAM*.

10.16 Network Architecture Components for OZIX

A comprehensive list of the OZIX networking architecture components is shown below. Asterisks mark those items that are still being studied for inclusion in OZIX.

[2] SUN is a trademark of Sun Microsystems Inc.

Communication Services

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- Internet Group Management Protocol (IGMP)
- External Gateway Protocol (EGP)
- Routing Information Protocol (RIP)
- Address Resolution Protocol (ARP)
- TP4 (ISO transport)
- ES-IS (ISO end node)
- UNIX domain sockets
- *Open Shortest Path First (OSPF) routing protocol

Communication Links

- Ethernet/IEEE 802.3

IBM Interconnect Components

Generic Components:

- DTF File Transfer
- DW 3270 Terminal Emulation
- 3270 Data Stream
- SNA Application Programming Interface
- APPC/LU6.2

Ported Components:

- Maintenance Operations Protocol (MOP)
- Common Trace Facility (CTF)
- SNAP - System Network Architecture Protocol (protocol analysis tool for SNA)

Access to OZIX via SNA:

- Medusa (LAT)

Application Programming Interface (API)

- X/OPEN Transport Interface (XTI)
- BSD sockets
- Network Computer System Remote Procedure Call (NCS RPC)
- Sun RPC (provided for Network File System interoperability only)

- DNA Session, OSI Session

File Store Services

- ftp (file transfer program)
- tftp (trivial file transfer program)
- rcp (remote copy program)
- Network File System (NFS)
- File Transfer and Access Management (FTAM)
- DECnet/OSI Copy Program (dcp)

Mail Services

- Simple Mail Transfer Protocol (SMTP)
- x.400 messaging

Name/Directory Services

- BIND/hesiod
- Distributed Name Service (DNS)
- x.500
- location broker for DEC RPC

Terminal Services

- Local Area Transport (LAT)
- TELNET
- dlogin, rlogin (remote login), rsh (remote shell)
- cterm
- OSI Virtual Terminal Protocol (VTP))

Network Management

- Common Management Information Protocol (CMIP), Common Management Information Service (CMIS), Network Command Language (NCL)
- Simple Network Management Protocol (SNMP)
- Internet Engineering Task Force (IETF) CMIP as recommended

Time Services

- DTSS - Digital Time Synchronization Services (DTSS)
- NTP - Network Time Protocol for internet

Boot Services

- Maintenance Operations Protocol (MOP) - Digital-proprietary
- BOOTP
- *BOOTPARAM

Security Services

- Kerberos
- Digital Authentication Security Service (DASS)
- Sun Secure RPC (for NFS interoperability)

CHAPTER 11

SYSTEM INSTALLATION AND SYSTEM DIRECTORY LAYOUT

11.1 Introduction

This chapter provides an overview to OZIX Version 1.0 system directory structure, system software installation, and layered application software installation. These subjects are covered in the same document because of their required integration to meet the goals of OZIX installations. This chapter first introduces the goals of OZIX software installation and then covers the system directory structure used to meet those goals. The discussion continues with a description of the system software installation and update procedure and concludes with layered application software installations and updates.

This chapter assumes the reader is familar with OZIX System Administration as described in the *OZIX System Administration Overview* [1]. Additionally, familiarity with the OZIX device and storage design, using the Attribute Based Allocation (ABA), aids the reader in understanding.

Throughout this chapter the term *installation* is used to refer to *software* installation. Hardware installations are not covered in this chapter.

11.2 Goals

The goals of OZIX system directory layout and installation are:

- Minimize service interruptions when software is installed or updated.
- Maintain system integrity during the installation period.
- Eliminate the need for coordinated releases of OZIX and layered applications because of version dependences.
- Simplify the installation procedure; the procedure should not preclude customer installability.
- Layered application installation procedures should be non-proprietary, so that third parties and customers may use these same procedures to install their applications.
- Installation of the base system should take less than one hour.
- Meet the priority 1 requirements as outlined in the *Preliminary OZIX Version 1.0 Serviceability Requirements Document* (SRD) [2].
- Provide compatibility with existing Digital ULTRIX third-party installation procedures.
- Provide remote, layered-application installations to and from OZIX.

11.3 System Directory Layout

OZIX system directory will follow the ULTRIX model as illustrated in Figure 29.

Figure 29: OZIX System Directory Tree

To Be Provided During Design

Additionally, the directory layout will support compartmentalization of log files as required by the SRD.

11.4 System Installation

The primary goal of initial system software installation is the overall time to install. To accomplish this the following basic steps are used to perform an initial system installation:

1. A minimal OZIX system is booted from the distribution media.
2. All necessary questions are presented to the user. Questions include node name, address, initial passwords, and system configuration data. On line help will be provided for all questions and defaults will be applied where appropriate.
3. Those parts of the system directory structure requiring read/write access by the running system are created and populated. This represents less than 15% of the total data manipulation required for system installation.
4. The remaining read-only data in the system directory structure resides on the distribution media. This represents the remaining 85% of data manipulated by the system installation procedure.
5. The minimal OZIX system used in steps 1 through 3 is terminated and the full OZIX system is started. The full system uses the read/write area created in step 3 above and the read-only data on the distribution media.
6. At this point a full OZIX system is available for full functional use. Although performance may be temporarily limited by the access speed of the distribution media, the system operates in a normal fashion, obtaining read-only data from the distribution media and applying updates to the read/write area created in step 3 above.
7. Since the distribution media may not have an acceptable I/O rate, the data in the read-only areas on the distribution media is transferred to a "real" system directory. During this time, the system is still fully functional.
8. After the data has been transferred, the distribution media is removed and system operation continues from the real system directory.

In general, system installation requires only one reboot of the system. Installation time is reduced to the amount of time it takes to build the read/write data on the system. The time it takes to transfer the read-only data, which accounts for about 85% of the data, is not included in the installation time since the system is fully usable while this transfer is taking place.

11.5 Application Installations

In order not to create another installation procedure with which applications must conform, the ULTRIX setld utility will be ported to OZIX. Refer to the ULTRIX documentation set for details. Applications conforming to setld software distribution technology will not have to be modified to install on OZIX.

In addition, remote software installation support will be provided by porting the ULTRIX Remote Installation Service (ris). Refer to the ULTRIX documentation set for details.

11.6 Internationalization

System installation and update are fully internationalized. If possible, some of the internationalization profile will be derived from settings in the hardware. If not possible, the installation procedure will have to query, in English, for the profile.

Internationalization of layered-application installation is limited to that which is provided by setld and each individual application's installation procedure.

11.7 Related Documents

[1] M. A. Ditto, *OZIX System Administration Overview*, Version 1.0, September 27, 1989.

[2] T. Baker, J. Bingham, M. Licata, D. Miller, T. Siebold, L. Stivers, *Preliminary OZIX Version 1.0 Serviceability Requirements Document*, Revision 0.9, August 4, 1989.

CHAPTER 12

PERFORMANCE ENGINEERING

12.1 Abstract

This chapter presents a high-level overview of the instrumentation and performance analysis methodologies for OZIX, and the services provided to accurately characterize its performance and the performance of applications it supports.

12.2 Introduction

In general, the extensive use of performance instrumentation and modern analysis techniques in parallel with software design is often overlooked by developers. More commonly, designs are completed and if unacceptable performance is observed during their testing, steps are taken to identify and correct the problem.

To meet the needs of the production systems market, however, a comprehensive performance strategy must be implemented from the beginning of the software design, rather than after the fact. Production systems markets require comprehensive instrumentation and performance analysis in order to provide the information necessary to enable effective capacity planning, configuration management, system management, performance characterization, and to support transaction processing in general.

This chapter presents our strategy for integrating performance analysis into the earliest phases of software design, generally, and into OZIX specifically. This chapter provides some background on OZIX performance methodologies, and then describes OZIX instrumentation and data collection. This chapter also describes the performance characterization tools for OZIX, and the OZIX modeling strategy.

12.3 OZIX Performance Engineering Goals

The Performance Engineering goal is to ensure that OZIX exhibits leadership performance when compared to systems offering comparable functionality and robustness. Our role is to define the metrics, develop the methodologies and tools, and support the designers in the achievement of this goal. Specifically, this translates into the following set of goals:

1. To provide OZIX developers, Independent Software Vendors (ISVs), and end users with the set of leadership tools and instrumentation services necessary to analyze and improve system and applications performance.
2. Establish and characterize clear and valid performance measurements that illuminate the differences in performance between OZIX-based systems and other systems whose operating system is of comparable scope.
3. To develop a leadership capacity planning strategy.

12.4 OZIX Performance Methodologies

This section provides an overview of how OZIX performance will be measured and characterized.

12.4.1 Workload Characterization

In order to assess OZIX component performance we need to have on hand a set of *standard* workloads with which we will study the contribution ratios of OZIX and OZIX components, and the equilibrium kinetics of OZIX-based systems.

We propose, initially, to acquire the following workloads (and benchmarks):

- The USAA, DuPont, Batelle, and Citibank workloads

  These workloads were chosen because of the detail in which they are described and their commercial nature. These workloads are sufficiently different, especially in their I/O characteristics‡ as to constitute a broad suite of complementary workloads.

- The Debit/Credit, Warehouse Inventory Control (WIC), and GE NFS benchmarks

  These Benchmarks are of little design interest to engineering. They command a high level of interest to the corporation because, for better or worse, customers make buying decisions based on the results of these benchmarks.

Using these workloads and benchmarks, we will apply the methodology developed for the scalable weighted application test (SWAT) kernels to generate synthetic and parameterizable performance kernels. The SWAT kernels are composed of two sets of kernels: The first is a suite of some 25+ synthetic routines, each of which is designed to exercise a specific attribute of a computer's CPU and memory complex. The second is a set of computational kernels derived from 5 different user applications. By running each suite, data is produced that may then be analyzed to determine those computer attributes that correlate most closely with application kernel performance. Technically, the methodology used to establish the sensitivity and correlation matrices, is general linear least squares analysis using singular value decomposition. This approach will enable us to more easily emulate a customer's computing environment; The load imposed by the customer's usage patterns on their systems.

As OZIX evolves, this approach should allow us to spend more time characterizing the performance of OZIX and less time acquiring and running workloads.

12.5 OZIX Instrumentation and Data Collection

12.5.1 Overview

The extent to which designers and users are able to critically evaluate the system performance of OZIX-based systems, and the extent to which such evaluations lead to meaningful and accurate measurements, constitutes a powerful incentive to prefer OZIX-based systems over the competition. To achieve this, OZIX needs a consistent, low-overhead, and comprehensive set of instrumentation and data collection services that applications and system software can employ to this end. To ensure that OZIX meets these requirements we will establish a consistent policy for instrumenting OZIX and its components, i.e., an *Instrumentation Architecture*. We will design a data collection facility, a low-level software component, that will enable the system to collect component and system-wide data; the data collection facility will conform to the entity management architecture (EMA) to provide consistency of access.

‡ Specifically, transfer size, request rate, physical locality, and file congestion

The instrumentation architecture must support two data collection methodologies—Event detection and sampling.

12.5.2 Event Detection

An *event* is a change in an object's state. *Event detection* is the process of dispatching and recording information regarding events.[1]. Thus, event detection is a technique to be used when it is necessary to know a sequence of events or the exact number of their occurrence over a given interval of time. Most capacity planning applications, for example, use event collection to produce performance and utilization traces to drive a model of the system in order to predict future requirements.

OZIX will employ *software probes* for event detection. A software probe is a recording routine dispatched upon the occurrence of an event. A *hook* is inserted into the code at a point such that when the event occurs control is transferred to the probe. After recording the event, and possibly other information, control is returned.

OZIX must have event detection capability carefully integrated into its component designs. If event detection is not used judiciously, or if the software probes are not placed with great care, system performance can be greatly degraded. Moreover, OZIX system management will be employed to control selection of event collection. Together these attributes ensure that event detection is implemented with the least cost to performance and access to event detection services is consistent.

12.5.3 Sampling

Sampling is a statistical technique usable whenever the measurement of all the data characterizing a set of people, objects, rates, or events is impossible, impractical, or too expensive. At regular intervals system- and/or component-specific counters are read. Sampling is commonly used to observe rates and frequency of events. It is also used to obtain a snapshot of the current state of the system at some particular point in time (i.e. number of users, amount of free memory remaining, etc). The process of data collection by sampling causes less disturbance to the system then event detection by software. It is less accurate then event detection, but can be made sufficiently accurate if the samples collected are reasonably large.

Sampling is used to measure the fractions of a given time interval each system component spent in its various states. The data collected during the measurement is subject to analysis to determine what happened during the interval, in what ratios the various events occurred, and how the different types of activity were related to each other.

12.5.4 OZIX Instrumentation Data Collection Facility

The OZIX Instrumentation Data Collection Facility *IDCF* is responsible for collecting performance and utilization data maintained by OZIX. The IDCF will provide data drawn from various levels of the system. Data collected by the IDCF will be categorized as follows:

- Subsystem-specific data
- OZIX-specific data
- Digital-specific data[1]
- Distributed-system specific data

[1] The point at which an event is detected is also known as a trace point. The process of event detection is commonly referred to as tracing

[1] Digital specific information pertains to specific Digital hardware or software. For example, Disk I/O statistics

The IDCF conforms to the common collection facility architecture being developed for all three of Digital's operating systems, ULTRIX, VMS, and OZIX. This will enable third-party vendors, ISVs and other groups in Digital to provide tools for understanding distributed system performance, dynamic load balancing, system configuration and distributed capacity planning. For more information see the OZIX Instrumentation Collector Functional Specification .

12.5.5 OZIX Instrumentation Services

As part of the instrumentation architecture OZIX will provide services for control of, and access to, OZIX instrumentation. The instrumentation services will conform to and employ OZIX system management. The instrumentation services will satisfy the DECxTP instrumentation requirements and will provide the foundation for the OZIX capacity planning strategy.

An example of some of the instrumentation services provided by OZIX:

- Access and control of counters for sampling (real-time) information
- The registration of probes for event detection
- Activation and deactivation of probes

12.5.6 OZIX Instrumentation Architecture

The following figure illustrates a conceptual, high-level overview of the OZIX instrumentation architecture.

The instrumentation architecture specifies the flow of data from the lowest levels of the OZIX system via manageable entities to the user. From a functional perspective the components are:

- The Management entities are instrumented components whose instrumentation is specified by their designer according to the OZIX Instrumentation Guide for Developers.
- The EMA backbone (including the Event Dispatcher) provides access to the instrumentation provided by the manageable entities.
- The OZIX Integrated Data Collection Facility (aka Data Collector) is responsible for collecting the data from the manageable entities.
- The Common Collector is an abstraction that provides a uniform mechanism by which to access operating system instrumentation. The services will be uniform across multiple operating systems (e.g., VMS, OZIX, ULTRIX, and OSF/1).
- Various data analysis and resource utilization applications can make use of the information collected by the Common Collector.
- The Presentation Management layer provides a set of presentation services (e.g., graphical visualization) to the user.

The Performance Group is working with representatives from VMS, ULTRIX, and DECcp, among others, to provide a mechanism by which performance and other data can be presented uniformly across a distributed, heterogeneous environment. Currently, the Common Collector will be supported by OZIX, ULTRIX, VMS, and OF/?.

Figure 30: The OZIX Instrumentation Architecture

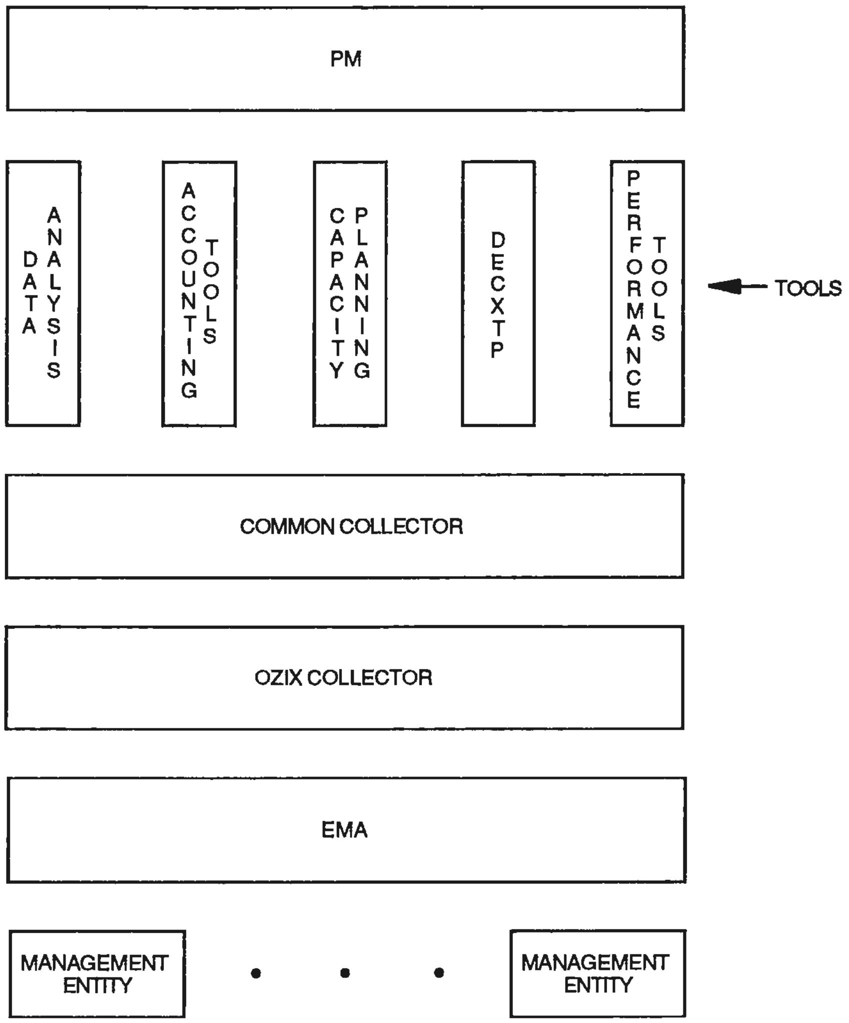
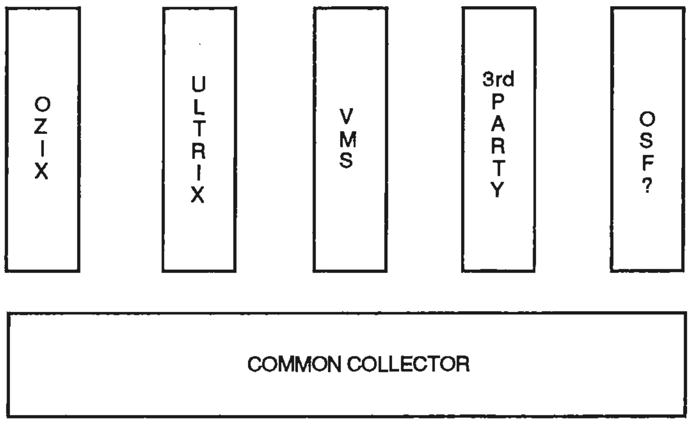


Figure 31: The Common Collector Facility



12.5.6.1 Security Implications

The OZIX instrumentation collector and the common collector will be designed in accordance to the OZIX B2 security guidelines.

12.6 OZIX Performance Characterization Tools

Initially OZIX will have a set of traditional UNIX utilities and commands including profilers and timing utilities, as well as tools that capture and present resource utilization information. Unlike UNIX, however, OZIX will incorporate a data collection strategy derived from an instrumentation architecture. This will enable Digital, third-party vendors, and even end users to build much better performance tools than are enjoyed by users and managers of other UNIX-based systems. Using this approach, we will be in a much stronger position to encourage and motivate the development of more comprehensive resource utilization monitors, sophisticated system management tools, and more accurate capacity planning applications. These, among others, fulfill critical requirements for success in the commercial production markets targeted by Digital for OZIX-based systems.

12.6.1 System Performance Characterization Tools

UNIX and its derivative operating systems offer a minimalist approach to providing tools to obtain system characterization measures. While the first release of OZIX will provide an identical set of commands and utilities, over time OZIX developers and users will be able to exploit the extensive instrumentation inherent in its design to produce a rich set of capabilities with which to measure system performance.

The problem with traditional approaches to measuring the system performance of UNIX-based systems is that there is no underlying policy governing the gathering of performance statistics. The tools (e.g., vmstat, netstat, etc.,) developed to extract these statistics are inconsistent, often incomplete, and inadequately understood in terms of the extent to which they affect the measurements themselves. They do not tell the user where the performance problem in the system is. It is left up to the user to run a collection of these tools and then make the determination as to where the performance problem exists.

OZIX, by contrast, will have a designed-in instrumentation strategy. This will help to ensure that the tools developed by Digital, ISVs, and end-users are consistent and meaningful. In addition, we will work with DECxTP to develop reporting and analysis tools which use advanced visualization techniques that will enable the user to diagnose and correct performance problems. To ensure compatibility with UNIX and its derivatives we will also provide the standard UNIX system activity tools and other standard functionality (e.g., vmstat, netstat, fstat, sa* etc.,).

In order to characterize a components contribution to system performance it must be instrumented. OZIX, and its components will be highly instrumented for this purpose. Moreover, the instrumentation will conform to an overall architecture that will make data collection consistent across OZIX-based systems and its components. In addition, and as indicated earlier, the IDCF will be provided. The instrumentation architecture and the IDCF which uses it will enable us to develop new and more appropriate tools for component performance characterization. Concurrent with the development of the instrumentation architecture and the design of the IDCF, we will produce a supporting document, *OZIX Instrumentation Guide for developers*.

12.6.2 Component Performance Characterization Tools

12.6.2.1 Elapsed Time Analysis

To fully characterize the performance of a component under evaluation (CUE), only an analysis of its elapsed execution time coupled with a detailed knowledge of its *execution profile* (see below) can yield the relationship between it and system performance. For example, a CUE may execute in N CPU microseconds. The actual elapsed time may be 10 times that, however, because it spends time waiting for resources held by other components. One of the problems with "UNIX" PC sampling is that the data it provides is not obtained in such a way as to indicate to the developer *what the system was doing while the CUE was waiting*. OZIX will provide services that will help a software developer determine where the system is spending its time, or what resources are being held, while a CUE is blocked. This is vital information to a developer wanting to enhance performance.

12.6.2.2 Execution Profiling

An execution profile of a CUE is a detailed time-ordered record of events detected during its execution. A typical execution profile sufficient for design optimization might contain:

- Instruction data - Path lengths, frequencies, basic blocks
- Elapsed time information
- CPU cycle usage
- Call graphs
- Register usage
- Memory references
- Istream references
- Lock statistics - Locks attempted, acquired/released, etc.
- Device I/O request data - Frequency, size, read or write, etc.

Generating the execution profile of a component is a necessary step in determining its impact on overall system performance.

Over the lifetime of OZIX it will be necessary to provide such services and support (e.g., documentation, training) as to enable developers to generate detailed execution profiles. Accordingly we will provide these services and a set of tools necessary for OZIX designers and other OZIX software developers to critically assess their component's impact on system performance. The services necessary to generate execution profiles and those which we will provide are:

- General event detection and collection
- *In situ* basic block coverage
- Non-invasive instruction tracing
- *In situ* I/O tracing
- Cycle counting

12.6.2.3 Coverage

Coverage analysis allows one to identify what sections of a CUE were executed. This allows the developer to identify dead regions of code that were never executed. We will provide tools for basic block coverage of both user and executive level software.

12.6.3 Initial Performance Characterization Tools

As previously stated OZIX will initially support the traditional UNIX tools. These tools are used by system administrators to determine how the OZIX system is performing, by system administrators and software developers to determine a components contribution to system performance and by software developers to characterize their components performance and to provide coverage information[1]. OZIX will initially provide the following tools:

- System characterization tools (these tools are also used in determining a components contribution to system performance)
  - SA* - System activity reporting tools
  - ipcs,pstat,vmstat,iostat,netstat - interprocess communication,system, virtual memory, I/O and network statistics
  - Instrumentation services - To provide control and access to OZIX instrumentation. (These are defined in the (*OZIX Performance Architecture Guide*)
- Component characterization tools
  - Pixie, pixstats - Pixie instruments a component, pixstats reports the results
  - Prof - PC sampling and coverage reporting tool
  - Time - timing information
  - ps - process status

These, among others, constitute the basic services necessary to obtain overall system utilization and computational capacity information - so vital for system management and capacity planning functionality. The quality of these tools will depend heavily on the instrumentation services provided by OZIX which is precisely why they are being incorporated as a part of its basic design. As it is the integration of instrumentation services and system tools that are traditionally referred to as software monitors, a key part of the OZIX strategy will be to work with third-party vendors to develop such monitors; perhaps with advanced visualization techniques that will enable even non-experts to characterize and manage system performance.

All the performance tools produced by the OZIX performance group will conform to the OZIX internationalization guidelines. Any new tools that require a command line or shell interface will conform to the OZIX command & shell style guidelines.

12.6.4 Capacity Planning

Capacity planning is the process of predicting and planning computing resources to meet business growth requirements or anticipated needs. OZIX will have an industry leading capacity planning strategy. That strategy is based explicitly on providing a leadership operating system for capacity planning applications. Leadership means that the OZIX services provided to capacity planning applications meet *all* of their instrumentation requirements. Moreover, for OZIX to be considered leadership in this area the capacity planning applications that run on OZIX must produce more accurate and more useful results than applications running on other operating systems. The quality of these particular applications and the value OZIX customers place on them are the final measure of leadership in this area.

[1] A component is some executable code (i.e. a program, an OZIX subsystem or shared library).

The fundamental requirements in providing a capacity planning application, and how OZIX will meet them are:

- Instrumentation

  Relevant data collection is the first, and in some ways, the most important requirement in capacity planning. The OZIX IDCF will provide capacity planning applications a comprehensive yet low-overhead mechanism to conduct event detection and other relevant statistics.

- Data analysis

  Data analysis routines within a capacity planning application make little use of operating system services. They do require, though, that the data collected via the operating system's instrumentation services are able to be abstracted into workloads that are meaningful (and valid) to the users. For example, "users per day", "number of terminals supported", and so forth.

- Modeling and Prediction

  A model of the system is then generated (usually employing a combination of simulation techniques for accuracy and analytic techniques for speed) and validated against the utilization data created during the data analysis phase. The anticipated workload is then parameterized and run on the model and the results are used to extrapolate to a new configuration.

  This is a critical phase of the capacity planner. The accuracy of the model is the principal measure of its quality. The best capacity planning tools on the market today are accurate plus or minus 30%. It is an OZIX goal to be accurate to within 20%.

  Configuration prediction will be based on transaction response time and/or system throughput as a function of the user's anticipated system load. For example, if the enterprise anticipates the number of users to grow by 20% per year over the next five years what additional capacity will the enterprise be required to purchase in order to maintain existing levels of performance?

- Reporting

  The reporting software must present meaningful results to different levels of personnel; This might include MIS managers, their technical staffs, and division, even CEO and CFO level executives. It is critical that OZIX not make it difficult or otherwise impede the generation of high quality, and easy-to-comprehend reports.

The implementation of the capacity planning architecture will involve work by OZIX developers, Digital' Capacity Planning Engineering Group, and most especially third party capacity planning software vendors. The requirements for the OZIX capacity planning strategy will be provided and driven by OZIX Performance Engineering with support from OZIX product marketing and other organizations as necessary.

12.7 OZIX Modeling Strategy - Overview

Our modeling strategy is to provide two distinct, but integrated levels of modeling support. The distinction between the two levels is made according to whether a *simulation* model or a *simulator* model is employed in any particular OZIX performance study. This is determined by the extent to which the execution of the system, or system component, must be explicitly represented. The two levels discussed here, which form the cornerstone of the modeling strategy are, the creation of both simulation and simulator models.

1. *Simulator Models*

Simulators are particularly useful for component performance characterization. A simulator is simply an abstraction of the hardware architecture implemented in software. Its principal advantages are, but not limited to:

- It can execute code in advance of the existence of real hardware.[7] This allows critical component performance studies to be conducted early in advance of the design and test phase of OZIX.
- Performance collection within a simulator is non-invasive. This means that a component under evaluation does not necessarily have to be instrumented in order to collect measurement data. Therefor the CUE runs as it would in a real environment.
- A simulator may be designed to produce performance statistics not practical in a real environment or without various types of hardware monitors (i.e., cache and/or bus monitors, instruction counts, cycle counts etc).

We will provide simulator support for OZIX developers by extending and/or redesigning existing simulators (e.g., Sable). This will be particularly important to developers working on early baselevel components.

2. Process-oriented Simulation

   Simulations are useful to OZIX developers for characterizing system performance (throughput, response time, etc.,) as a function of operating system design. A process-oriented performance simulation is a software program in which processes are created which make requests of other processes (e.g., OZIX components) and/or hardware platform components (models of the CPU, memory subsystem, etc.) for resources, scheduling, and other services.

   We are prepared to provide and support process-oriented simulation programs, tools, or support that focuses explicitly on network and I/O performance problems.

12.7.1 OZIX Modeling Goals

Our overall goal is to provide modeling tools, services, and support to OZIX component and system designers. This overall goal is our way of serving a shared overarching goal, i.e., to insure that OZIX achieves the very highest performance that technology, product constraints, and engineering skill is able to deliver.

The goals of our modeling effort are:

- To enable the characterization of the performance of individual OZIX subsystems, and the OZIX executive without the need for actual hardware.
- To enable the characterization of the performance of OZIX as a function of the underlying hardware architecture.

12.7.2 OZIX Modeling Strategy

For high level OZIX performance studies, we will design a simulation model of the OZIX executive. This model may be designed from existing models when one exists to do the job (i.e. a network model). Initially we will use this tool study such questions as:

- System performance prediction as a function of the layout of executive subsystems.
- System performance prediction as a function of I/O software design and the underlying I/O component configuration.

[7] A compiler capable of generating target hardware instructions is, of course, required.

- System performance prediction as a function of network interconnection and flow-control strategies.

The simulation model will be written in C, will employ the CSIM run time library.

For performance characterization studies requiring the explicit execution of subsystem code, we propose to build or extend existing architectural simulator. The simulator will provide an execution environment for detailed characterization of subsystem performance (as well as program development/test, fault generation/error recovery, and portability). The simulator will provide the following features:

- Target instruction set and cache layout
- Multiprocessing
- 32 and/or 64-bit virtual addressing
- Extensive instrumentation
  - Accurate CPU timing (via cycle counting)
  - Exact instruction counts
  - Basic block counting
- Integration with Ladebug, Prof, and other commonly used UNIX tools.
- Instruction set independence

Visualization tools will be constructed where appropriate.

CHAPTER 13

SOFTWARE QUALITY AND TESTING STRATEGY

13.1 Overview

This chapter describes:

- Each component of the quality and testing strategies and the role the component plays in the overall strategy
- The motivation, requirements, goals and nongoals behind the extensive and aggressive software testing strategy to ensure we are building a quality product
- The components and capabilities that must be delivered by various engineering groups to support the quality and testing strategies

13.1.1 Motivation

The product goals for OZIX demand that the system provide 7x24 availability, conform to industry standards (POSIX, OSF, X/OPEN, etc.), and be extremely reliable, secure, and robust in order to be competitive in the production systems marketplace. Quality, reliability, and security must be designed into the system in addition to thorough testing of the completed system for the necessary levels of robustness.

In defining the testing strategy for the system, we have attempted, where possible, to utilize component strategies that provide the greatest *coverage* for the lowest cost. This chapter defines a composite testing strategy, specifying a variety of testing techniques and tools to be employed to obtain the highly reliable, robust, error-free products required.

A thorough, effective test system allows us to make corrections or enhancements to the system and subsequently run the test system against the updated software:

- To ensure that no existing functions were broken as a result of the change
- To permit development of new tests to cover the changes for future development
- As a regression test to ensure that the same problem isn't reintroduced into the system in the future

13.1.2 Software Quality and Testing Strategy Requirements

The quality and testing strategy depends on:

- Software designs and implementations which are high-quality, well-engineered and well-integrated

  Without well-integrated, well-engineered designs, we waste precious resources tracking down logic, design and integration errors that should have been caught during the design and implementation processes.

- Testing methodologies and tools to:
  - verify that the designs are implemented as specified
  - verify where possible that all erroneous inputs and data, singly or in combination, are detected and handled properly
  - exercise and load the system to determine the interactions of components and the effects on the system of depleting resources

13.1.3 Software Quality and Testing Strategy Goals

The goals of the quality and testing strategy are to:

- Identify methodologies and procedures to introduce quality, reliability, security, usability, and maintainability into the system before implementation and testing is begun
- Define methodologies, tools and guidance for developers building and executing test sets
- Build test systems which:
  - Confirm that the system behaves correctly under specified and adverse circumstances
  - Provide a structure into which unit test procedures, error verification procedures, and so on can be easily added
  - Eliminate the reoccurrence of "fixed" problems through regression tests
- Provide a measurement of the code coverage attained by the test system
- Provide a method to track bugs and fixes, and to enhance the test suite to prevent the same bug from being reintroduced into the system

System testing is an ongoing activity, not a point event. The testing strategy must provide plans for the life cycle of OZIX.

13.2 Software Quality and Testing Strategy Components

The components of the software quality and testing strategy are described in the following sections.

13.2.1 System Specifications

The design of OZIX is being done in a formalized manner, with the end result being a functional specification, an interface specification and a detailed design specification for each component of the system.

The formats for all of these documents and the document review process is outlined in the *OZIX Software Development Procedures*. The following paragraphs summarize the documents and their review procedures.

The functional specification is a high level summary of the component functionality. The functions it describes will map to the interface specification for details, to the detailed design specification for implementation, and to the code for function performance.

After the functional specification for a component has been completed, it must pass two reviews. The area review (performed by a designated team of developers known as area reviewers) determines if the specification fits within the overall vision for the project, delivers the required capabilities, and is complete.

The primary review is performed by team members, area members, and members from other areas. The specification is reviewed for its technical content, completeness, and quality of the interfaces presented.

After the functional specification passes primary review, it is put under change control and is made available to the entire software group.

The interface specification for each system component describes all external interfaces to the component. All externally visible data structures are also documented. Using this information, developers of other components are able to successfully write code that invokes this component.

The interface specification must pass one review which may be combined with the detailed design review at the discretion of the project leader.

The detailed design specification for a component contains the logical module designs. This chapter is reviewed by the project team for design validity and completeness, and put under change control. It is primarily used by the project team during the implementation phase to produce the code for the documented component. It is also valuable during project maintenance, for training of new project members, for creating user documentation, and for communicating the design to external groups.

13.2.1.1 Benefits

Producing the functional, interface and detailed design specifications is an enormous amount of work. However, the benefits of this effort far exceed the effort invested:

- The comprehensive design specification and review process help ensure the quality, consistency, coherency, completeness, and thoroughness of the design
- The detailed designs are used to build *white box* tests for integration testing and base level testing of system components
- Over time, as new team members are brought into the software group, they are able to read a formal document describing the system, rather than being forced to learn the design by tediously reading the code
- The documents can be used by layered product developers to gain a deeper understanding of the system, reducing the need for interaction with members of the OZIX development team
- The documents can be used to maintain the code and to aid in design of future releases of OZIX
- The documents can be used for the development of training materials for product support

13.2.2 Code Reviews

As the development work turns to coding, we must ensure that the code is of the highest quality, and is consistent across the product. One way in which this will be accomplished is via code reviews. The following paragraphs summarize code reviews. More details and pointers to other documents on code reviews can be found in the *OZIX Software Development Procedures*.

A code review is a good mechanism for information exchange among developers. It is also a good method to catch subtle bugs that a developer may be too close to the code to see. The earlier a bug is caught, the less expensive it is to fix.

The goals of a code review are to check:

- Adherence to coding standards
- Correct solution of the problem
- Correspondence of code to design

- Correct use of language features
- Maintainability of the code
- Understandability of the code
- Efficiency of the code

The coding standards for the code developed on this project are defined in the *OZIX Naming Standards and C Coding Conventions* document.

Although it is ideal to have every line of new code reviewed, it is usually impossible to do so. All code which makes up the *trusted computing base (TCB)* for *B2* security must be reviewed. Interface routines, asynchronous routines, frequently called routines, exception handlers, time critical routines, hardware error routines, and complex routines should be reviewed. It is up to the project leader to determine which code will be reviewed if all the code cannot be reviewed. Complex routines should be walked through line by line to detect logic or coding errors.

Enhancements made to ported code should also be reviewed. The project leader and developer decide on the amount of code needed for the review to provide adequate context for the enhancements. Enhancements to ported code should follow the style of the original code unless a new module is created. In this case the code in the new module should follow the *OZIX Naming Standards and C Coding Conventions* as much as possible.

13.2.2.1 Benefits

Effective code reviews done early in the development process promote a higher quality product by:

- Pinpointing problematic code
- Ensuring that all code is developed using the same coding style. This promotes more readable software, and over the long term, aids in reducing maintenance costs.
- Providing a better understanding of effective coding constructs
- Providing a review of the validity of the implementation and correspondence with the design
- Familiarizing developers other than the author of the module with the code
- Providing feedback to other developers, and potentially, the compiler group
- Detecting algorithmic errors in complex modules at earlier stages of development, reducing time lost later in the project

13.2.3 Unit and Integration Testing

Developers are responsible for the unit testing of components and testing the integration of components. The purpose of this testing is to give the developer and project leader reasonable assurance that the implementation is error free.

The Software Test Team develops templates for unit tests and routines to generate reusable test cases to assist the developers in writing comprehensive unit tests for their code.

The test cases are developed using *equivalence partitioning* and *boundary value analysis* to ensure the greatest test coverage at the least overall cost. These test selection methods are described in detail in G. J. Meyers' *The Art of Software Testing*. Test cases are structured to detect sensitivities in parameter presence, range of legality, and combinations of problems.

This framework provides a mechanism for individual developers to easily add unit test code to the test system if appropriate. The tests developed during unit and integration testing may be added to the test system to be executed on all new system base levels to ensure that properly functioning code continues to execute correctly and that expected errors are detected.

Over time, the tests developed during unit and integration testing may be added to the regression test suite to increase test coverage. The Software Test Team builds the basic test case library, regression test suite, and develops a "how-to-use cookbook" to allow developers to add tests more easily to the test suite.

13.2.3.1 Benefits

The regression/unit test framework:

- Relieves the developers of the tedium of building the test framework, allowing them to concentrate on the tests, hopefully allowing more tests to be built
- Provides a flexible, extensible structured framework on which a complete unit test system can be built

13.2.4 Executive Testing

Testing is performed on the OZIX Executive code (*nub* and executive *subsystems*) by both the developers and the Software Test Team. Tools used for building unit tests are utilitized for building base level tests. Many unit and integration tests for the executive become part of the base level regression test suite.

Building the base level test suite is an iterative process that involves constructing the tests, evaluating the results, correcting any problems found, and enhancing the test suite for additional coverage. Tests are also added at each base level to test all new functionality added to the executive for that base level.

Time is allocated in each system build to permit execution and verification of the base level test suite.

13.2.4.1 Benefits

The testing of OZIX executive code:

- Exercises the executive code more thoroughly than can be done at a higher level
- Provides better coverage than tests executed at user mode
- Finds problems earlier than normal in the development cycle of the system

13.2.4.2 Nub Routines Test Suites

Test suites are being developed for the nub routines. System level hooks are used in combination with debuggers and hardware simulators to run these tests before application level functionality is available on the system.

Tests for the executive interface routines are developed using both white box and *black box* approaches. The code is examined and the tests are written to cover as many code paths as possible. Boundary conditions and error handling are stressed in particular.

Since routines change and new ones are added at each baselevel, the tests are also appropriately modified by the developers and the Software Test Team.

13.2.4.3 Executive Subsystem Tests

Subsystem tests are written by the developers with assistance from the Software Test Team. These are developed using both white box and black box approaches. A method to simulate errors which are not easily generated for testing is being investigated to assist in the thorough testing of error paths in the subsystem.

All security testing, device-related error testing and any required fault insertion is performed by the developer. Where possible, tests are added to the regression test suite. The Software Test Team is working with the appropriate groups to develop a detailed strategy in this area.

13.2.5 Application Level Testing

Application level testing involves testing of the *API* and application subsystems including commands, utilities, system services, and library routines. Porting the *ULTRIX Test Executive (UTE)* to OZIX for this testing is being investigated. Test suites developed by UEG for ULTRIX are modified for testing OZIX specific features as necessary. We are investigating the feasibility of modifying the POSIX conformance tests to run under UTE.

13.2.5.1 Benefits

The user mode test suites:

- Promote building a maintainable, extensible highly-structured suite of tests for runtime routines
- Automatically test ported code, because the tests for this code will also be ported
- Share testing code with the ULTRIX/OSF group

13.2.5.2 Commands and Utilities Test Suite

A test suite is being developed to test the commands and utilities available on OZIX. The test suite is based on the UTE commands test suite with modifications and enhancements made for OZIX specific features.

13.2.5.3 API Subsystem and Library Test Suite

Outside of the normal unit and integration testing done for these components, a test suite is being developed to test the external interfaces to OZIX. This test suite is based on the UTE system services and functions test suites with modifications and enhancements made for OZIX-specific features.

13.2.5.4 DECwindows/Xwindows Test Suite

Currently an Xlib validation suites exists for ULTRIX, but no automated procedures exist for capturing and comparing screen images. An OSF version of *DEC/Test Manager (DTM)* is planned for the same time frame as OZIX FRS, but it will not be available during most of the OZIX development. A method to record and playback user input on all types of workstations is in the process of being developed by OSG. This tool (currently named Xigor) depends on the Xtrap server extension being installed with the Xserver. The script files generated by Xigor are planned to be in the same format as needed by the OSF version of DTM making it a viable tool to use during OZIX development although we will have to add screen image capture and comparison to it.

13.2.5.5 AIA Routines Test Suite

A test suite for all OZIX-supported *AIA* routines is being developed to run under UTE.

13.2.6 Base Level Regression Testing

Tests are run on each base level of the system immediately after it is built. These tests are a combination of unit and integration tests from various components, user mode tests, and system integration tests. The base level build is not considered ready for general use until all tests complete successfully.

The regression tests are run only after an initial stable base level has been developed and tested using the unit and integration tests for the OZIX executive.

13.2.6.1 Benefits

The benefits of base level testing are:

- Verify correct system build
- Early identification of system integration problems
- Release of stable base level software

13.2.7 Problem Tracking

Starting with base level regression testing and continuing throughout product development and delivery, all bugs are tracked using a problem reporting system which is expected to be integrated with the project CASE environment, if at all possible. If the bug was not identified through testing, then, if possible, a test is added to the test suite to ensure that the bug doesn't reoccur. The correspondence between the bug report, the bug fix in the code, and the test are maintained. Changes to documentation and code (whether they result from bugs or not) must be done via that change proposal mechanism described in the *OZIX Software Development Procedures*.

13.2.7.1 Benefits

Problem tracking:

- Identifies what problems have been fixed and which still exist
- Builds a better regression test suite
- Ensures that fixed bugs do not reoccur

13.2.8 Test System Coverage Analysis

In order to verify the effectiveness of the test system coverage, we must be able to determine the extent of the system code exercised by the test system. The profiling tools developed by the performance group will be used when possible.

The Software Test Team analyzes the coverage information obtained during the execution of the full test system. They then develop a prioritized list of untested code paths which are candidates for fault insertion testing, or determine what automatic test can be developed to cover the paths.

The goal is to use the test system to exercise 80% of the code.

13.2.9 Full-time and Stand-alone Usage of the System

As soon as the developing system is capable, software system builds, other test suites, and application suites are run frequently on in-house development systems in order to maximize system usage and to determine system behavior when running with little idle time. We anticipate that this load should exercise the system and provide beneficial feedback without affecting users very much.

A moderate amount of testing is necessary on dedicated machines for areas such as power failure, hardware errors, system crashes, low-level system debugging, system and configuration management, etc. This testing is scheduled and users have access to stand-alone machines for this purpose in order not to impact users on development systems or be impacted by system activity.

13.2.10 OZIX System Installation Verification Procedure

The OZIX system installation verification procedure (IVP) is a required component of the software kit. It is executed as part of the system installation procedure to ensure that the system has been properly installed and consists of the System Exerciser running under the On-line Diagnostic Monitor (see Section 13.2.21) and a selected subset of software tests.

13.2.10.1 Benefits

The software IVP:

- Reduces support costs by ensuring that a correct installation was accomplished
- Increases the probability that the installation will be trouble-free
- Provides a product acceptance test

13.2.11 Field Test

Field test is the engineering development process by which the total hardware/software/support system is exercised. It provides a generally friendly environment in which the system is utilized heavily in a nonproduction mode to shake out problems not detected or impossible to shake out in the in-house environments. Field test sites are selected based on criteria specified by engineering, to fulfill specific needs such as applications used, hardware environment, and so on.

Field test is an engineering activity. It is not intended for marketing or sales use. It is not a mechanism by which early systems can be placed into customer installations, unless this placement serves the needs of engineering.

Preliminary field test requirements and plans will be issued as part of the Phase 1 documentation.

The feedback loop, typically through a Quality Assurance Report (QAR) system, will be implemented such that there is no additional time or levels of bureaucracy introduced between the submission of the problem report and its receipt by engineering, permitting engineering to respond promptly to the reports.

13.2.12 Software Documentation Testing

We must ensure that the software documentation accurately and completely describes the software/hardware product that is shipped. This is done by extensive review of the documentation by the engineering group, by internal users, and by our field test sites. It is planned that several of the field test sites will be contacted during field test to determine the accuracy and usefulness of the documentation.

The individual writers are responsible for ensuring that their manuals have an effective review and are reviewed by the responsible engineers. Each manual has a key developer responsible for the technical accuracy of its contents and technical signoff.

13.2.13 Stress/Load Testing and Subsystem Interaction

Stress and load testing aids in defining the operating envelopes and performance ranges of the systems. The OZIX System Exerciser is used to load the system in a controlled manner. Subsystem interaction is stressed by running multiple copies of the same tests for various applications.

13.2.14 Configuration Testing

The Software Test Team is responsible for determining the minimum hardware configuration for OZIX. Testing of OZIX will be performed on some reasonable number of other configurations, some of which will include network devices.

13.2.15 Network Testing

A large amount of network testing is done in unit/integration testing and in testing applications which access the network. It is expected that TCP and DECnet certification will be performed by NaC.

13.2.16 Interoperability Testing

The interaction between OZIX and other systems (Workstations, VMS, ULTRIX) will be tested by the SVT lab. A separate interoperability test plan is being developed.

13.2.17 System Management Testing

There are two areas of concern for System Management testing. The first is testing the DECwindows interface. Some tools may be available in the future for DECwindows testing (see Section 13.2.5.4). Second, the current System Management design involves managing multiple machines. There are currently no tools to accomplish distributed testing.

13.2.18 Conformance/Compliance Testing

Conformance/compliance test suites are being run for POSIX 1003.1 and 1003.2, OSF, X/OPEN XPG3, ANSI C, Motif, Xlib and *NFS*. All of these test suites are being provided by outside organizations. It is expected that the development groups for SCS, MSCP, and LAT will obtain any internal conformance tests for these areas. The Software Test Team provides help in running of tests if needed. Network certification is discussed in Section 13.2.15.

13.2.19 Usability Testing

Usability testing is performed on components which are providing a user interface (System Management, On-line Diagnostic Monitor, System Exerciser, On-line Documentation, Messages). A separate usability plan is being developed.

13.2.20 B2 Certification

OZIX V1.0 will not be B2 certified, but should not preclude certification. OZIX V2.0 or some release between V1.0 and V2.0 will be certified. Testing for OZIX V1.0 takes security into account and is performed with the goal of having to duplicate as little as possible during B2 certification. The test plan may be refined as we obtain a better understanding of B2 certification.

13.2.21 Hardware Testing Strategy

The hardware testing strategy for systems running OZIX consists of several components, some of which are directly linked to the development of OZIX.

13.2.21.1 Hardware Testing Overview

It is expected that hardware systems on which OZIX runs are provided with a fairly extensive set of ROM-based diagnostic programs, a minimal set of standalone loadable diagnostic programs to cover any parts of the system boot path not covered by ROM-based diagnostics, and a relatively large set of online diagnostic programs.

The development of diagnostic programs is not a part of OZIX development. The following aspects of a hardware testing strategy *are* a part of the OZIX project:

- Development of the Online Diagnostic Monitor.
- Development of a system exerciser.
- Development of a system error logger and an error log report generator.

13.2.21.2 Online Diagnostic Monitor

The Online Diagnostic Monitor (ODM) provides the user interface and the controlling mechanisms for running online diagnostic programs. Online diagnostic programs are used to test or exercise system hardware and/or software without taking the system offline, thus causing a minimum amount of disruption to system availability.

13.2.21.2.1 ODM Architecture

The following are major highlights of ODM's architecture.

- It is designed to be independent of the hardware platform on which it runs.
- It provides a single, consistent, internationalized user interface for all ODM-compliant online diagnostic programs. Two versions of this interface are provided. One version is a DECwindows implementation, and the other is a command-line implementation capable of running on any type of user terminal.
- It provides a wide variety of user options, such as the ability to:
  - Run a single diagnostic program.
  - Run multiple diagnostic programs, either serially or in parallel.
  - Select the devices to be tested, the tests to be run on the devices, the length of time the tests should run, and whether to stop or continue when errors are reported.
  - Using default values for user options, simply select the devices to be tested and issue a "start" command.
  - Execute "script" buffers of ODM commands.

- It provides a set of runtime services to diagnostic programs for creating internationalized error report displays and user prompting messages.
- Users do not need to know diagnostic program filenames.
- Users do not need to know hardware device addresses.

13.2.21.2.2 Online Diagnostic Program Architecture

Following are architectural highlights for online diagnostic programs that run with ODM:

- They can be ported to all OSF-compliant environments since they perform I/O operations using X/OPEN-defined system calls.
- They can make use of diagnostic functions provided within OZIX device drivers, if non-portability is acceptable for selected diagnostics.
- They perform user terminal I/O via ODM services (which use underlying POSIX-defined system calls).
- They are internationalizable.

13.2.21.3 System Exerciser

The On-Line System Exerciser (OX) is a system exerciser that applies software loads to the system and exercises the hardware/software interface. This exerciser runs under the On-Line Diagnostic Monitor (ODM). ODM provides a standard interface to all hardware diagnostics including OX.

OX is utilized:

- By the network, I/O, and file systems groups to exercise their hardware/software interfaces
- By the performance group to apply specific loads to the system while taking performance measurements
- As one piece of the acceptance test for the system at customer installation

The use of OX provides:

- Exercisers for disk I/O, file system, network I/O, CPU, memory management, and tape I/O
- A method by which the I/O, network, file system, and performance groups can load the system in a controllable manner for their own testing
- A way to stress test the OZIX system software and hardware during development to aid in discovering software errors
- A tool for manufacturing to use to perform Stage III checkout and burn-in testing
- A tool for Customer Services to use to verify the installation of new hardware or software at the customer site and to troubleshoot system problems

13.3 Overall Quality and Testing Strategy

When implemented, this strategy demonstrates that the software product reliability and quality goals have been met. The overall strategy is to:

- Conduct design and code reviews
- Define the test plan
- Specify the component tests
- Plan for resources, including manpower and hardware and software
- Develop and document the tools and test suites needed to support the test plan
- Execute the test plans
- Feed back results as appropriate
- Track bug reports and fixes, improving test suites as needed and as resources permit

This strategy is a repetitive, adaptive process which continues over the life of the product set.

Many of the results of the test plan execution are data that can be used for both engineering and to form the basis for marketing documentation.

13.4 Risks

- Internationalization—The impact of internationalization and multi-language support on testing is unknown at this point.

13.5 Associated Documents

1. The *OZIX Software Development Procedures* which describes the process of developing components of OZIX.
2. The *OZIX Naming Standards and C Coding Conventions* which describes the project coding standards.
3. The *OZIX Performance Engineering Overview* which describes the performance engineering work for OZIX.
4. The *OZIX Security Certification Plan* which describes the strategy to obtain a B2 security rating.
5. The *OZIX Usability Plan* which describes the strategy for usability testing of OZIX.
6. The *Interoperability Test Plan* which describes the strategy for interoperability testing of OZIX with other systems.

GLOSSARY OF TERMS

***-property**: The property that a higher clearance level subject may not write to a lower sensitivity level object.

7x24 Operation: This means non-stop operation through 7 days of a week, and 24 hours of a day.

Access controls: Algorithms, architectures or hardware that control the access of subjects to objects.

Access stacks: A sequential set of modules that limit the access rights of subjects to objects. The access validation in OZIX is composed of one or more access stacks.

Access validation component: The OZIX component responsible for making access decisions.

ACSE: See *Association Control Service Element.*

Active State: A state through which objects are actively transitioning, i.e., arriving and departing

Address Resolution Protocol: The Internet protocol used to dynamically bind a high level Internet address to a low level hardware address

Agent: Provides operation dispatching services to manageable objects. At the core services layer, all management applications request operations to manageable objects via the agent.

AIA: Application Integration Architecture

Analytic Model: A software program that solves a series of explicit mathematical equations. Many queuing models are formulated as mathematical equations which are then solved. Often termed Back-of-the-Envelope calculations.

API: See *Application Programming Interface.*

Application Programming Interface (API): A layer of software that provides a standard-conforming programming interface for use by applications.

ARP: See *Address Resolution Protocol.*

Association Control Service Element: That portion of the OSI Application Layer common to all OSI Applications. It controls extended connections (e.g., *associations*) between OSI applications.

Assurance of Resources (AOR): An OZIX feature which assures that one or more users cannot prevent users of higher priority access to the resources of the system.

Attribute-based allocation (ABA): A storage management architecture, which uses......

Authentication: The process of verifying the identity of a user. Authentication is especially important in a network environment where a message must be shown to have been issued by a particular user.

Authentication: A method of proving the identity of a user.

Availability: The probability that a system will function correctly at an instantaneous point in time.

Available: A level of service that withstands a single failure in each type of hardware component through redundancy, and restores service quickly in the event of software failures through fast reboot.

B2 Certification.: A level of security defined by the Department of Defense DOD characterized by structured protection

Backbone network: The collection of subnetworks responsible for forwarding data between other interconnected subnetworks.

Base containers: This is a container, whose physical configuration table enumerates only a single media in its entirety.

Basic Block: A region of the *CUE* that can be entered only at the beginning and exited only at the end

Black box testing: A method of software testing in which the tester is unconcerned about the internal behavior and structure of the program. Test data are derived solely from the specifications.

Block device: A random-access mass-storage device that conducts I/O operations using blocks or chunks of data.

Block device interface: Through this interface, data is transferred in multiples of sectors. Data transfer occurs between the media and the system buffer. Block device interface is accessible directly through the appropriate device special files.

BOOTP: A Internet protocol specified by RFC1084. This protocol is used by systems during their boot phase to obtain information about their environment. This protocol is not used by any vendors today and is currently inadequate for diskless workstations.

Boundary value analysis: A method of selecting test input and output data based on using values at the edges of equivalence classes.

Broadcast: A packet delivery system that delivers a copy of a given packet to all hosts attached to it.

Character device: A device that provides either a character-stream oriented I/O interface or, alternatively, a raw interface.

Character or Raw device interface: Through this interface, data can be accessed directly from the device. Data transfer occurs directly between the media and the user buffer. This interface is typically used by applications that have intimate knowledge of the data structure on the disk or tape device.

Clearance level: The part of a subject's label that determines what sensitivity levels of data he may access

Client: As part of a client-server model, the client is the system that initiates communications and requests service from a server. In the NFS system, the client is the component that accepts requests from users for file access and sends those requests to a file server. See also *server*.

CMIP: See *Common Management Information Protocol.*

CMOT: A double acronym for CMIP (Common Management Information Protocol) over TCP/IP. An Internet standard to support the CMIP protocol over TCP/IP connections.

Collector: Collector: collect and filter data. A collector gathers the gauged data from a set of probes or other collectors. It may then filter/process/archive them and pass them to other collectors or presenters

Common Management Information Protocol (CMIP): A protocol used for distributed network management. There are two flavors within DECnet/OSI, the DNA version, based on a draft of the second, the OSI standard for network management. (See also CMOT.)

Common Management Information Services (CMIS): The services needed to support CMIP.

Compartments: A grouping of related data.

Component Contribution Ratio: A high level representation of the sensitivity of overall system performance to a component's execution

Compound container: Container whose configuration table enumerates other compound containers, subcontainers or base containers. All the elements in a compound container are included in their entirety within the compound container. The components that are included to form the compound container are called constituent containers.

Confidentiality: Protecting data from unauthorized disclosure.

Connected endpoint (for datagram services): An endpoint that has pre-specified its destination port and address. Once set, the address cannot be changed for the life of the endpoint.

Container: This is a virtual storage device. It is uniquely named, persistent, and provides an array of data blocks for read and write operations. A container consists of container metadata and container data.

Core Services: The client/server orient services provided by the OZIX management backplane. Client applications can use the services provided by the agent, event dispatcher, event sink, and MIR servers.

Coverage analysis: A method of determining which lines of code were executed during a specific test.

Covert storage channels: An unusual method of passing information that violates the system's security policy

CTERM: A virtual terminal protocol specific to DECnet/OSI.

CUE: Component Under Evaluation is the collection of software that has been instrumented for performance characterization

DAP: See *Data Access Protocol.*

DASS: See *Digital Authentication Security Service.*

Data Access Protocol: The DNA Access Protocol. It is a proprietary remote file access service within DECnet/OSI.

Dataspace: A dataspace is a logical unit of storage used by higher levels of software, such as the file system or a data base application, to read or write data. Each dataspace has a unique, invariant name

Dataspace archive: The storage management space used solely for the purpose of archiving dataspaces. Storage manager migrates inactive dataspaces from dataspace migration domain into dataspace archive.

DCP: See *DECnet Copy Program.*

DECnet Copy Program: The application on OZIX and ULTRIX that implements the client side of the DECnet/OSI DAP protocol.

DECnet/OSI: That part of OSI embraced by the applicable DNA standards, plus extensions specific to DNA.

Digital Network Architecture: The collection of standards and architectures that comprise the DECnet specification.

Digital Time Synchronization Service (DTSS): A Digital product that provides distributed time services.

Discretionary Access Control (DAC): Access control based on the identity of the accessing subject

Discretionary model: The algorithm which defines DAC

Distributed Authentication Security Service (DASS): A Digital product that provides distributed authentication services based on RSA public key encryption techniques.

Distributed Name Service (DNS): A Digital product that provides distributed naming services.

Distributed System Security Architecture (DSSA): An architecture under development within Digital to provide strong security within a distributed system.

Distribution Services: The lowest layer of the management backplane. The distribution services insulate the high layers from the specifics of the management protocol and transport services used.

dlogin: The application on OZIX and ULTRIX that implements the client side of the DECnet/OSI CTERM protocol.

DNA: See *Digital Network Architecture.*

DNS: See *Distributed Name Service.*

DSSA: See *Distributed System Security Architecture.*

DTM: DEC/Test Manager

DTSS: See *Digital Time Synchronization Service.*

Elapsed time analysis: Analysis of a *CUE* that reflects the effects of the system (such as paging) on that particular run.

EMA: Enterprise management architecture. This is the architecture that describes system management of enterprise environment.

End node: A node within a DECnet/OSI network that does not involve itself with routing decisions.

Enforcement: Compelling obedience

Equilibrium Kinetics: The study of state transition kinetics - A classical methodology designed to study the dynamics of distributed systems. Found commonly in network performance studies

Equivalence partitioning: A method of partitioning of test input data into classes such that a test using a representative value of a class is equivalent to a test using any other value in the class.

Error: Occurs when a portion of the system assumes an undesirable state. If the error is observed at the output of the system by the user, it is considered a failure.

Event: A report that describes an asynchronous state change.

Event detection: The process of detecting an event. This is commonly referred to as tracing

Event Dispatcher: A server at the core services layer that provides manageable objects with a mechanism to post event.

Event Filter: A definition of the type of event to which the event subscriber listens.

Event Sink: A depot of events to be read by subscribers of events. Each event sink receives the event type defined by its event filter.

Execution context: The logical address space which can be referenced by an executor

Executor: A family of threads with the same security and accounting identities sharing the same set of SECs.

Executor model: The architectural model that defines an executor.

Exported filesystem: An entire local filesystem, a directory tree, or a single file that has been exported by a file server. An exported filesystem may be mounted by an NFS client.

Extended Services: Currently the highest layer of the management backplane architecture. These services are to be purely object-oriented and are TBD.

Failure: Occurs when the delivered service deviates form the specified service

FAL: See *File Access Listener*.

Fault: The underlying cause of an error, not necessarily physically identified

File Access Listener (FAL): The application on OZIX and ULTRIX that implements the server size of the DECnet/OSI DAP protocol.

Filesystem: An implementation defined disjoint tree of directories. A filesystem must be mounted into the system namespace before it can be accessed. See *namespace*. This is different from *file system*.

File system: The software that provides file services.

File Transfer and Access Management: The service and protocols defined by ISO for remote file access.

Fragmentation: The technique used by IP to break an outbound transport data request into segments that can be handled by the subnet-dependent device used to transmit the data. An identification number is placed in fragment for the destination to use in reassembling the datagram.

FTAM: See *File Transfer and Access Management.*

Gate: The linkage mechanism used to move from an SEC in one subsystem to an SEC in another subsystem.

Gateway: A computer that attaches to two or more networks and routes packets from one to the other.

Hooks: Involves the insertion of a trap code, jump instruction or some other type of dispatch mechanism. The process of enabling detection in this manner is commonly referred to as hooking

Identity stacks: a structure which records the execution history of a thread

Instrumentation: The process of adding additional code at various points in a CUE to record information

Instrumentation services: The infrastructure and interfaces available for obtaining instrumented data

Integrated Management: The same management framework for system, network, and application management in a distributed environment.

Integrity: The correctness of a program or of the state of the system; the protection against destruction or corruption of data

Integrity Access Control (IAC): Controlling access based on the amount of trust placed on code.

Integrity Level: A measure of how much faith one has in the correctness of a piece of code, the value of a piece of data.

International capabilities: The functions of an international product that support the languages and conventions of more than one locale.

Internationalization: The process that includes the development of an international product and the localization of the international product for delivery into worldwide markets.

Internet address: The 32-bit address assigned to hosts that want to participate in the Internet using TCP/IP.

Internet Protocol: The Internet standard protocol that defines the Internet datagram as the unit of information passed across the Internet and provides the basis for the Internet connectionless, best-effort packet delivery service. The Internet Protocol suite is often referred to as TCP/IP.

IP: See *Internet Protocol.*

IP datagram: The basic unit of information passed across the Internet. An IP datagram is to the Internet as a hardware packet is to a physical network. It contains a source and destination address along with data.

ISO: International Standards Organization.

Kerberos: A distributed authentication service based on private key encryption techniques, from project Athena.

Label: A combination of a security level, an integrity level, security categories and integrity categories.

LAN: See *Local Area Network*.

LAT: See *Local Area Transport*.

Least privilege: Allowing an executor to have no capabilities beyond those required to perform his duties

Level: Sometimes used as a synonym for label (see *label*). Also may refer to part of a label, either the security level or integrity level.

LM segment: An abstract vector of memory pages

Local area network: A physical networking medium designed for use within a limited geographical area. This term is frequently used to represent a class of high-bandwidth broadcast datagram networks, such as Ethernet.

Local Area Transport (LAT): A transport service protocol based directly on top of Ethernet, commonly used for terminal-host connections.

Locale: The local environment in which a product is used, including language, dialect, keyboard, data input and display conventions, collating sequence, and other attributes that directly affect how users interact with the product.

Localizable software: Software that can be modified to suit particular locales.

Localization: The process of adapting an international product to suit the language, conventions, and market requirements of a particular locale.

Location Broker: A form of name service specific to RPC, used to locate a specific remote service.

Logical memory segment: See *LM segment*

Maintenance Operations Protocol (MOP): A special low-layer protocol within DNA used for maintenance operations such as booting and diagnostic services.

Malicious: intending to cause harm

Manageable Object: An object that registers its class definition with the management backplane. Management operations performed on the object are issued from a management application, through the management backplane, and to the object.

Management Application: Applications that request the services of the management backplane to monitor and/or control manageable objects.

Management attributes: Part of the Enterprise Management Architecture (EMA) model. A named characteristic of a management object that may be changed.

Management Backplane: The component of system administration that interconnects manageable objects and management applications.

Management Control Language: A semi-extension of NCL.

Management Information Repository (MIR): Contains class definitions of all manageable objects registered with the management backplane.

Management object: Part of the Enterprise Management Architecture (EMA) model. Each manageable entity defines what its objects are. An object is the target of the following operations: create, get, set, action and delete.

Management Service RTL: The run-time library (RTL) that management applications use to access the services of the management backplane.

Management Services: The set of services provided by the management backplane. This includes the services provided by the Extended, Core, and Distribution Services.

Mandatory Access Control (MAC): Controlling access based on the sensitivity level of data and the clearance level of users

MCL: See *Management Control Language*.

Message: Short descriptive text associated with a condition value.

Migration domain: This represents a set of containers. The storage manager uses the space to migrate active dataspaces, to enact storage management policies.

Model: Any abstraction of the real world. With reference to computer systems the term *model* is usually applied to a description of a computer system, component, or software program. Such descriptions are often codified into software programs that mimic, or otherwise behave as, that which they describe.

MOP: See *Maintenance Operations Protocol*.

Mount point: An empty directory on which another filesystem is attached.

Multicast: A technique that allows copies of a single packet to be passed to a selected subset of all possible destinations. Some hardware supports multicast by allowing a network interface to belong to one or more multicast groups. *Broadcast* is a special form of multicast in which the subset of machines to receive a copy of a packet consists of the entire set.

Multicast groups: A group of hosts that are logically grouped together. They share the same IP multicast address. Each host in the group receives a copy of a multicast datagram addressed to the group.

Multi-level devices: Devices which can contain data of more than one sensitivity level

Multilingual software: Software capable of supporting user interfaces in more than one natural language at a time.

Mutexes: A type of lock which assures MUTual EXclusion

Name service: A service that keeps track of names of objects known within a network, as well as attributes associated with each name, such as location or address.

National Computer Security Center (NCSC): The organization responsible for certifying the security of systems

NCL: See *Network Control Language.*

Need to know: The principle which states that a person should have access to only that data required to perform his job

Network Control Language (NCL): The application and its interface specification used to implement the client side of the DNA CMIP protocol on some Digital operating systems.

Network Interface Definition Language (NIDL): The specification language for Digital's RPC interface compiler.

Network layer: The third layer of the OSI Reference Model, responsible for relaying information between the source and destination transport entities.

NFS: Network File System

NIDL: See *Network Interface Definition Language.*

Node: A logical data origination or termination point within a network, usually associated with a physical computer system.

Nub: The most basic portion of the OZIX system which provides subsystems with primitive functions.

Object: Defined by its class definition (class name, attributes, operations, and events) and realized by creating new instances of its class.

Object Class: Definition of the object, which includes the definition of attributes, operations, and events.

Object Instance: The actual object itself. It is the instantiation of the object class.

Octet: Eight bits.

Open system: A system who's services and interfaces are known and available to outside parties.

Open Systems Interconnection (OSI): The term Open Systems Interconnection (OSI) qualifies standards for the exchange of information among systems that are *open* to one another for this purpose by virtue of their mutual use of the applicable standards. (ISO 7498)

Orange Book: Department of Defense Standard DoD 5200.28.STD, Dec 1985: the DoD Trusted Computer System Evaluation Criteria—the document that defines the requirements of the certified classes.

OSF: Open Systems Foundation

OSG: Open Systems Group

OSI: See *Open Systems Interconnection.*

Package: A set of subsystem procedures

Partitioned model: A model for replicating services to withstand failures. In this model some of the available resources act as primary providers of the service, and others act as passive secondaries. The secondaries are synchronized with the primaries such that they can resume service after failure of a primary.

Phase V DECnet/OSI: DNA has been designed in phases. Phase V represents the current architecture in which OSI protocols have been embraced.

Physical memory management: The system's management of its physical memory resources

Portal: A software mechanism providing a path of communication between communicating peers through a network based on incompatible protocols.

Port number: The number used to represent an address at the Internet transport level.

POSIX: Portable Operating System Interface for Computer Environments. A collection of IEEE standards.

Presentation: The sixth layer of the OSI reference model, responsible for negotiation and transformation of syntax between cooperating applications.

Priority: An assigned measure of the relative importance of a task

Profiling: The process of collecting and analyzing data that describes the run-time behavior of a *CUE*.

Proxy ARP: The technique used to allow one machine, usually a gateway, to answer ARP requests intended for another by supplying its own physical address. By pretending to be another machine, the gateway accepts responsibility for routing packets to it.

Rainbow books: The set of DoD documents issued by the National Computer Security Center, which defines the requirements and interpretations of certified computer systems. This collection of documents includes the Orange Book.

RARP: See *Reverse Address Resolution Protocol.*

Raw-device interface : The character-device interface for block-oriented devices such as disks and tapes. Through this interface, data transfer occurs directly to and from user's data buffers.

Reliability: The probability that a system will function correctly over a specified period of time

Reliability level: A measure of how much faith one has in the correctness of a piece of code, the value of a piece of data

Remote Procedure Call (RPC): In general, a style of network communication that is modeled after procedure calls. Clients make requests of servers that return replies. A particular implementation of RPC called Sun RPC is used by NFS for its communication.

Resources: Available source of supply or support that can be drawn on when needed.

Reverse Address Resolution Protocol (RARP): The Internet protocol that a diskless machine uses at startup to find its Internet address. The machine broadcasts a request that contains its physical hardware address and a server responds by sending the machine its Internet address.

Ring brackets: A two element value which controls the level of trust required to read or write an object

Ring levels: The level of trust of executable code

Robustness: Insensitivity to incorrect operation, inputs or errors.

Routing: The forwarding of data from the source to the destination along a series of one or more intermediate paths.

RPC: See *Remote Procedure Call.*

Sampling: Sampling is the process of examining event counters at some defined interval

SEC: The virtual memory accessible to an executor in a subsystem.

Secure RPC: A version of Sun RPC that uses the Data Encryption Standard (DES) for authentication. See *authentication.*

Security: Preventing the unauthorized disclosure of data

Security classes: Sets of objects subject to particular access controls

Security kernel: A relatively small set of code which implements a system's security

Security model: The algorithm controlling access to data based on sensitivity levels

Security policy: The rules defining how subjects may access data based on sensitivity levels

Sensitivity Level: A measure of how important the confidentiality of a data item is

Server: In a client-server model, the server is the component that receives requests from clients, executes the requests, and returns any results. In the NFS system, the server is the component that executes file requests on behalf of clients. See *client.*

Session: The fifth OSI service layer, defined by the Basic Reference Model to provide a common means of synchronizing the transfer of data.

Session control: A layer specific to DECnet/OSI that integrates the selection of protocols and use of underlying layers into the operating system on which it is running.

Simple Network Management Protocol (SNMP): An Internet-defined management protocol.

Simple security policy: The property that a subject may not read data which is more sensitive than his own clearance level

Simulation Model: A software program in which a computer system's hardware and software designs have been highly abstracted. Simulation models are used to answer performance questions that can not be answered with analytic techniques.

Smart card: An extremely small, self-contained processor used for authentication

SNA: See *System Network Architecture.*

Software Probe: The code that gets control upon the detection of an event. Probe: meter/observe events and status. By putting gauges in proper places and activating them allows the metering of events or status

SPC: See *Subsystem Procedure Call*

State Equilibrium: Two or more active states are said to be in state equilibrium when they are at steady state *AND* objects are actively transitioning between them

State Transition Rate: The change in the number of objects in a given state over some arbitrary interval of time

Steady State: A steady state is achieved when an active state's state transition rate equals to zero

Subcontainers: This represents a partition of a container.

Subjects: Active entities, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state

Subnetwork: A topologically connected set of network entities sharing a common addressing technique. For example, an Ethernet or X.25 component of a larger network.

Subsystem: A specially designated body of code in the OZIX system. Code and data in one subsystem are protected from code and data in other subsystems by memory management. Multiple executors in the same subsystem are separated from one another and cannot communicate.

Subsystem Execution Context (SEC): See *SEC*

Subsystem Procedure Call (SPC): A mechanism for passing execution flow from one subsystem to another in a controlled manner. SPCs pass through security gates (see *gate*).

System Network Architecture (SNA): IBM's collection of networking architectures.

System State: System state information provides the state transition information. Some examples might include waiting on IO completion, executing, waiting on a lock

TCB: See *Trusted Computing Base*.

TCP: See *Transmission Control Protocol*.

Thread: A flow of execution

Time service: A service that provides accurate time and date information within a subnetwork.

Timing Analysis: The process of providing timing information, elapsed and cpu for the duration of an event. An event can be the execution of a particular command, program or a *CUE*.

TP4: See *Transport Protocol 4*.

Translation: The process of rendering information presented in one natural language into another natural language.

Transmission Control Protocol (TCP): A protocol in the Internet protocol suite. This protocol provides a reliable, virtual-circuit transport service.

Transport: The fourth OSI service layer, defined by the Basic Reference Model as providing transparent transfer of data between peer entities which relieves them from any concern with the detailed way in which reliable and cost effective transfer of data is achieved.

Transport Protocol 4: An implementation of ISO transport limited to operating at the fourth class of service as defined within the ISO Transport Service specification. An informal term.

Trojan Horse: A program that surreptitiously performs some set of actions while performing an alleged primary action.

Trusted Access Control (TAC): Access control based on the trustworthiness of code. In OZIX, this is based on ring brackets

Trusted Computing Base (TCB): The portion of the system software that is trusted to properly handle data at multiple security levels at the same time.

Unconnected endpoint (for datagram services): An endpoint that specifies its destination port and address for every transmitted datagram.

Upper layers: The common term used to reference the OSI Session, Presentation, and Application service layers.

User Presentation: A management application with user presentation services, which is visible to system administrators.

UTE: ULTRIX Test Executive

Vendor Assistance Program (VAP): A service provided by the National Computer Security Center to assist system developers in producing certifiable systems

Virtual memory model: The model that defines the relationship of an SEC to physical memory

Virtual Terminal Protocol (VTP): The service and protocols defined by ISO for remote terminal emulation.

VTP: See *Virtual Terminal Protocol.*

WAN: See *Wide Area Network.*

White box testing: A method of software testing in which the tester derives test data from an examination of the program's logic.

Wide area network: One or more physical networking media designed for use over an extended geographical area. This term is frequently used to represent communications over leased telephone lines or X.25 networks.

X.25: A CCITT recommendation defining services and interfaces to public packet switching networks.

X.400: A CCITT recommendation for services and interfaces between electronic mail transfer agents.

X.500: A CCITT recommendation for interfaces and services to a directory service.

X/Open Transport Interface (XTI): A transport service interface that is independent of any specific transport provider. It is defined by the X/Open Group.

XTI: See *X/Open Transport Interface.*