PATHWORKS: PC Integration Software

Digital Technical Journal

Digital Equipment Corporation

Volume 4 Number 1

Winter 1992

The *Digital Technical Journal* is published quarterly by Digital Equipment Corporation, 146 Main Street MLO1-3/B68, Maynard, Massachusetts 01754-2571. Subscriptions to the *Journal* are $40.00 for four issues and must be prepaid in U.S. funds. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Orders, inquiries, and address changes should be sent to the *Digital Technical Journal* at the published-by address. Inquiries can also be sent electronically to DTJ@CRL.DEC.COM. Single copies and back issues are available for $16.00 each from Digital Press of Digital Equipment Corporation, 1 Burlington Woods Drive, Burlington, MA 01830-4597.

Digital employees may send subscription orders on the ENET to RDVAX::JOURNAL or by interoffice mail to mailstop MLO1-3/B68. Orders should include badge number, site location code, and address. All employees must advise of changes of address.

Comments on the content of any paper are welcomed and may be sent to the editor at the published-by or network address.

ISSN 0898-901X Documentation Number EY-J825E-DP

Book production was done by Digital's Database Publishing Group in Northboro, MA.

Cover Design
The red and blue threads woven together in our cover design represent the many PC clients and server systems that are integrated in a network environment by the outstanding "thread" of PATHWORKS software. PATHWORKS software for the integration of PCs over LANs and WANs is the featured topic in this issue.

The cover was designed by Kathryn Cimis of the Corporate Design Group.

Contents

Editor's Introduction

Jane C. Blake
Editor

The integration of personal computers in a network environment is the subject of this issue of the *Digital Technical Journal.* The software products that bring about this integration are known collectively as PATHWORKS and are derived from Digital's Personal Computing Systems Architecture. The engineering challenge for developers was to integrate a variety of client (PC) and server systems—DOS, Windows, OS/2, Macintosh, VMS, and ULTRIX—and to ensure that the intricacies of the meshing of these systems remained transparent to PC users.

In the opening paper, Alan Abrahams and David Low provide background for the papers that follow by describing the technical aspects of the various hardware and software platforms, physical networks, and protocols that had to be addressed by PATHWORKS developers. They also present an overview of the PATHWORKS components which allow PC users to access network resources.

Among the capabilities PATHWORKS enables, PC access to files on server systems is one of the most important for users. Two file servers, one for VMS and another for ULTRIX, were developed for this purpose. A paper on the development of the first of these, written by Ed Bresnahan and Siu Yin Cheng, contains an architectural overview of the VMS file server. The authors also detail the mapping done to bridge the differences between DOS, OS/2, and VMS operating systems. In a related paper, Phil Wells describes performance improvements made in version 4.0 of the file server which were achieved by optimizing the transport interface and the data buffering algorithm. He discusses the analysis of server performance for various interface models, the implementation of the algorithm in the VMS server, and test results.

Like the VMS file server, the PATHWORKS software for ULTRIX systems integrates PC clients with a server system on a LAN. However, as Anthony Rizzolo, Beth Brewer, and Martha Chandler explain in their paper, a multiple process model was chosen rather than the single process used in the VMS file server. The authors give their reasons for this different approach as part of a general discussion of the server design and implementation.

The network is key to the exchange of data in the PATHWORKS environment, and as is the case for the server software, multivendor systems must be addressed to ensure smooth integration. Mitch Lichtenberg and Jeff Curless describe how Digital has extended Microsoft's LAN Manager across a LAN or a WAN by using the DECnet transport protocol as the transport layer. In addition, they present the reasoning behind the design of the transport component for DOS and OS/2 products, and review steps taken to reduce memory usage and improve performance.

Further details on the integration of DECnet and LAN environments are provided in the paper on two network virtual device drivers for the Microsoft Windows environment. As Andy Nourse explains, these drivers manage DECnet and NetBIOS operations and enable the Windows operating system to support peripheral devices, memory resources, and software applications. Andy first gives readers background on the Windows operating modes, and then describes the development of the two virtual device drivers.

A significant new application in the PATHWORKS family, called eXcursion, brings together the capabilities of X Windows, DECnet, and the Microsoft environment, resulting in the display of both Windows and X Windows on the same screen. Dennis Giokas and Andy Leskowitz present the integration philosophy behind the display server and the implementation of the server architecture. They also relate how designers approached the mapping of the windows in the X and Windows environments.

The issue concludes with a paper by Chris Methot on capacity modeling of PATHWORKS client-server workloads. Chris describes a queuing analytical model used to understand resource consumption on the server and the special modeling process required in the client-server environment. The paper works through a specific example of the model's identification of bottlenecks in the system.

The editors thank Star Dargin and Carnel Hoover for their help in preparing this issue.

Jane Blake

Biographies

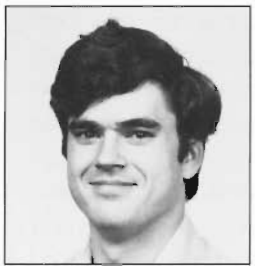
Alan Abrahams Alan Abrahams is a consultant engineer in the Personal Computing Systems Group Technical Office. He develops management and security strategies for integrating PCs into enterprise-wide networks. Alan joined Digital in 1982 and designed and implemented the PRO/Communications package. Since 1985, he has been the architect responsible for integrating Microsoft's LAN Manager into Digital's PCSA and helped design Digital's NetBIOS emulation and remote boot of MS-DOS systems. Alan received B.S degrees in computational and statistical science and in mathematics from the University of Liverpool.

Edward W. Bresnahan Senior software engineer Edward Bresnahan has been developing the PATHWORKS for VMS software since joining Digital's PCSG Server Engineering Group in 1988. He is currently responsible for the design and development of a high-performance data cache to be used in future PATHWORKS server products. Prior to this, he was a co-op student at General Electric Company and at Charles Stark Draper Laboratory. Ed holds a B.S.C.S. (1988, honors) from Northeastern University and is pursuing an M.S.C.S. part-time.

Elizabeth A. Brewer Beth Brewer is a supervisor in the PCIE Server Development Group—Open Systems. Beth served as project leader for the PATHWORKS for ULTRIX version 1.0 product as well as the principal architect and implementor of the PATHWORKS for ULTRIX administration process. She also worked for the PCIE Client Development Group—PC DECwindows. Beth joined Digital in 1987 after receiving a B.S. in mathematics with a minor in computer science from the University of Massachusetts at Lowell.

Martha A. Chandler A senior software engineer in the PCIE Server Development Group—Open Systems, Martha Chandler was project leader for the PATHWORKS for ULTRIX version 1.1 product. She designed and implemented the management interface for the PATHWORKS for ULTRIX server. Prior to this work, Martha maintained MS-Windows terminal emulation for the PCIE Client Development Group. Before joining Digital in 1988, she received a B.S. in mathematics with a minor in computer science from the University of Massachusetts at Lowell.

Siu Yin Cheng Since joining Digital in 1987, Siu Yin Cheng has worked on server software in the Personal Computing Systems Group. As a senior software engineer, she is responsible for the design and development of the server configuration utility for future PATHWORKS products. Siu Yin designed and developed the server collector process to extract performance data from the file server; she also worked on server development. Prior to this, she led the system testing of PATHWORKS server V2.0-2.2. Siu Yin received a B.S.C.S. (1987, honors) from Brown University.



Jeffrey R. Curless As a principal software engineer in the Personal Computing Systems Group, Jeff Curless worked on the OS/2 data link driver and on the PATHWORKS token ring implementation. He is currently developing a new configuration utility to support the future direction of the PATHWORKS product set. Since joining Digital in 1986, he has contributed to the development of PATHWORKS software under both the DOS and OS/2 operating systems. Jeff holds a B.S. in computer science from the University of New Hampshire.

Dennis G. Giokas Dennis Giokas is the group technical lead for PCSG's Network Client Engineering and the engineering manager for its New User Interface Group. His primary responsibility is technical lead for the next generation of the PATHWORKS client. Prior to this work, Dennis contributed to PC DECwindows development. Before joining Digital in 1984, he was employed by Arco Oil & Gas and The Foxboro Company. Dennis holds a B.M. (1974) from the University of Massachusetts at Lowell, an M.M. (1976) from the New England Conservatory, and an M.S.C.S. (1989) from Boston University. He has two patents pending.

Andrew T. Leskowitz A principal software engineer in the PCSG X Server Development Group, Andy Leskowitz is the project leader for the eXcursion display server. Since coming to Digital in 1987, he has contributed to various X development projects and designed the PATHWORKS LANSESS component. Andy's prior experience includes engineering positions at Datatrol, The Foxboro Company, and Raytheon Company. He has a B.S. (1976) in biology from Swarthmore College. Andy has applied for a patent related to his X server development work.

Mitchell P. Lichtenberg Mitch Lichtenberg is a principal software engineer in the Personal Computing Systems Group. He is responsible for the design and implementation of the PATHWORKS network client transport architecture and for various other aspects of Digital's PATHWORKS PC integration products. Before joining Digital in 1986, he was employed by the Xerox Palo Alto Research Center as a software engineer in the Xerox Artificial Intelligence Systems Division. Mitch holds a B.S. (1986) from Worcester Polytechnic Institute.

David A. Low David Low is a consultant engineer in the Personal Computing Systems Group. Since joining PCSG in 1988, David has worked in a variety of advanced development tasks involving PC networking technology. He is currently concerned with assessing approaches for pen-based computing and wireless PC networking. David has an A.B. in mathematics and an M.A.S. in computer science from Boston University. He is a member of AAAS, IEEE, and ACM.

Christopher E. Methot Chris Methot has been analyzing client-server performance since joining Digital in 1986. He has worked in performance characterization of LAVc systems and has contributed to VAX Performance Summaries. Currently, he supervises capacity/performance engineering in the Personal Computing Systems Group. In addition to developing the PATHWORKS client-server modeling process, his group is developing a standard performance test for Macintosh servers and has benchmarked many of Digital's hardware servers. Chris holds a B.S. (1967) in industrial design from the University of Cincinnati.

Andrew W. Nourse Principal software engineer Andrew Nourse has worked on network software for the PATHWORKS and DECnet-DOS products for the past six years. He developed Microsoft Windows and non-Windows networking applications, libraries, and drivers. Prior to this, he wrote network utilities for DECSYSTEM-20, DECsystem-10, and RSTS/E products. Andy received a B.S. in electrical engineering and computer science from the Massachusetts Institute of Technology in 1974 and joined Digital in 1976.

Anthony J. Rizzolo A principal software engineer in the PCIE Server Development Group—Open Systems, Anthony Rizzolo designed and implemented the PATHWORKS for ULTRIX file server process. He also designed the data link and port driver layers for the PATHWORKS for DOS product. Prior to this work, Tony was a member of the Internal Software Support Group and the TOPS-10 Engineering Group, where he designed and implemented the data link layer for the KLNI Ethernet adapter. Tony joined Digital in 1981. He received a B.S.E.E. from Stevens Institute of Technology.

Philip J. Wells Phil Wells is the PATHWORKS server architect and is responsible for coordinating the design and implementation of the PATHWORKS server products. In previous positions at Digital, Phil worked for Corporate Telecommunications designing Digital's internal network, the EASYNET, and helped support data centers and networks while in the Internal Software Services Group. Phil joined Digital in 1976 as a computer operator in the Corporate Data Center.

Foreword

Joseph A. Carchidi
Group Engineering Manager,
PC Integration

In the 1990s, a major shift is occurring in personal computing, from isolated, individual work on desktops to work in groups whose members are located throughout an enterprise. To support this important change, Digital has developed a family of products, called PATHWORKS, that enables personal computer users to make the shift from the standalone machine to the network environment and the resources of larger computer systems.

The roots of PATHWORKS were in place as early as 1980. Digital's engineering management recognized that a significant part of the growth in the computer industry would be redirected from minicomputer to microcomputer products. As the 80s progressed, we learned from our experience in personal computer hardware development and from the direction taken by the growing and highly competitive microcomputer market that industry standard-based products were more important than unique technologies; that is, open systems, comprising standard devices and interconnects, were what customers wanted, not more proprietary systems.

Digital's VAXmate personal computer, introduced in 1987, was built on the industry standard model. Moreover, it offered something no other PC offered at that time: the VAXmate had the network built in. With foresight, engineering management determined that our microcomputer business would tie to our long-standing strength in building networks. Our strategy thus changed from a focus on hardware development to the development of microcomputer software.

The critical question then asked—and the one that lead to PATHWORKS development within Engineering—was whether to provide customers with an upgrade path similar to those of competitors in the PC LAN business at that time, i.e., file and print services, or a network environment that embraced the primary technologies used by customers, i.e., a complete set of networking applications that included file and print services, mail, X servers, and terminal emulators. The strategy that took hold was the latter; we would develop a broad set of products that recognized customers' investments in a range of personal computer and network software. Unlike other single-product PC LAN offerings, this set of products would be engineered to couple large server systems based on CISC and RISC technologies with the primary microcomputer systems and would support operation over a local or wide area network. Furthermore, the mapping between the disparate systems would have to be transparent to users, and without concessions on performance.

This chosen strategy, of course, was not the easier of the two to implement. One of our initial tasks was to select which operating systems to support among the many microcomputer operating systems available in the market. We decided to define the scope of our early development work by supporting the most widely popular personal computers, which are those based on the DOS, OS/2, and Macintosh operating systems. Another important decision was the choice of a network transport that would serve as the basis for the interconnection of the systems selected. We selected Microsoft's LAN Manager software as this transport. MS-NET, the predecessor to LAN Manager, had the advantage of being network transport independent, thus allowing us to utilize the DECnet network to extend the PC LAN software to a wide area network.

In the papers in this issue, you will read about some of the extensive work that has been accomplished since we first embarked upon this software effort. Engineers have designed and implemented file servers and network transports that allow PCs to access files, applications, storage, and print services on the larger VMS and ULTRIX server systems. Further, a PATHWORKS application, called eXcursion, brings together the X Window System, the Windows environment, and the DECnet network. The effect is to link X—so important to users of UNIX systems—with the PC DOS system environment. These combined efforts represent a hallmark in Digital's progress toward open, heterogeneous computing.

Our achievement in the Personal Computing Systems Group has been our steady progress toward providing customers the open computing environment they need. The breadth of our product offering has taken on clear definition within the last year, and we will now begin the work of adding depth to the PATHWORKS product set. The possibilities for future developments are truly astounding. Looking ahead five years from now, client workstations will have the power of supercomputers, and the dramatic progress in parallel computing will bring additional opportunities for data sharing and application developments which are in embryonic stages today. Our challenge in software engineering will be to make all these systems work together in a well-integrated, easy-to-use, well-deployed computing environment.

Alan Abrahams
David A. Low

An Overview of the PATHWORKS Product Family

As the number of personal computers continues to grow, so does the demand for networking products and services to allow these PCs to share networked resources. Digital's Personal Computing Systems Architecture enables the integration of PCs into Digital's enterprise-wide network systems. The software products developed using this architecture are referred to as the PATHWORKS product family. PATHWORKS products support a variety of PC platforms and operating systems, and accommodate different physical networks and transport and service protocols. This flexibility allows PC users to access resources outside their PC environment, such as remote files, printers, databases, and electronic mail.

When the IBM Corporation introduced its first personal computer in 1981, few could have foreseen that by 1992 millions of PCs would have been sold worldwide, radically changing the computer market in the process. The term PC usually implies an Intel 80x86 family or a Motorola 68000 series processor, sized to fit under a desk or smaller and commonly priced under $5000. The low price has helped to fuel an explosive growth in the number of hardware products and software applications available for PC platforms. PCs are now ubiquitous and represent the largest class of networked computers.

Even before the introduction of the PC, small computers were being networked together to share data and hardware resources. In 1990, as many as 40 percent of the installed PCs were networked.[1] By 1994, an estimated 75 percent of the increasing number of PCs will be linked together with products from many networking vendors. These vendors provide services that commonly include transparent access to remote files, printers, databases, and electronic mail.

Digital Equipment Corporation is a worldwide leader in networking services. Since 1986, we have been developing the Personal Computing Systems Architecture (PCSA) to meet the growing needs of PC client-server applications in local and wide area network systems. Many technical obstacles were met and overcome in the design and development of PC integration products. The PATHWORKS product family, derived from PCSA, reflects the diversity of Digital's customers' needs and environments. PATHWORKS software products support a variety of PC platforms and operating systems, and accommodate different physical networks and transport and service protocols.

To help the reader comprehend the scope of the PATHWORKS offerings, we begin this paper with a basic discussion of PC hardware and software, followed by information about the various protocols used in PC networking. We then describe how Digital's PATHWORKS product set allows integration of PCs into network systems.

PC Hardware

This section describes the PATHWORKS Intel and Macintosh client platforms and introduces related PATHWORKS services.

Intel Platforms

The most popular operating systems in the world, IBM's PC-DOS, Microsoft's MS-DOS, and Microsoft Windows, are designed to take advantage of the features of the family of Intel chips that includes the 8086, 80286, i386, and i486 microprocessors.

The 80x86 memory architectures have evolved from 16-bit addressing with implicitly referenced 64-kilobyte segments in the 8086 processor, to 32-bit addressing with a paged virtual memory in the i386 or higher processors. Recent Intel processors have features previously associated with minicomputers. The i486 chip, for example, has an integrated floating-point processor, instruction and data caches, and hardware support for multitasking. This range of processor capacity highlights a major concern of the designers of Digital's PATHWORKS products, i.e., how to efficiently accommodate the range of differing functionality found in the installed Intel-based PCs.

Although this PC market has had little de jure regulation, IBM's market presence has shaped the de facto interface standards. The industry standard architecture (ISA) system bus and the video graphics array (VGA) display technologies are examples of such standards.

The most common system bus, the ISA bus, provides 16-bit data access to a 24-bit (i.e., 16-megabyte) address space. Physical and electrical interface conventions have been established and thousands of interface boards are available. IBM introduced the ISA bus and later developed the Micro Channel Architecture (MCA) bus, which provides 32-bit data access to a 32-bit (i.e., 4-gigabyte) address space, automatic bus sizing, and accelerated data transfer mechanisms. The MCA bus is not compatible with the ISA bus. Consequently, a number of manufacturers other than IBM joined forces and devised the extended ISA (EISA) bus, with features analogous to those of the MCA bus. Even though Digital's PCs use either the ISA or EISA bus, we support our customers' MCA bus machines through software and peripheral device offerings.

Graphical user interfaces (GUIs) such as the one provided by the Microsoft Windows software are becoming the rule rather than the exception. IBM's color graphics adapter (CGA) display was an early standard at 320 columns by 200 rows and a range of 4 colors. VGA is a more recent standard, with variants that can generate a screen up to 1024 by 768 in 256 colors. There is no widely accepted display standard beyond VGA, and it may be sufficient for manufacturers of innovative display technologies to provide device drivers for transparent use by Microsoft Windows applications. For example, the PATHWORKS eXcursion for Windows display server, which implements the X Window System protocol and operates in the Microsoft Windows environment, uses the display drivers supplied with the Windows software. The eXcursion server thus leverages any new display technologies with which Windows drivers are supplied. However, the standalone DOS-based X Window System servers supplied with the PATHWORKS software must be modified to use a new display technology.

Network interface cards (NICs) provide access to local area network (LAN) systems. NICs that adhere to ISA, MCA, and EISA standards are available from dozens of manufacturers for many networking topologies. Digital manufactures NICs for thick, thin, and twisted-pair Ethernet connections. PATHWORKS products support the Network Datalink Interface Specification (NDIS) and thus accommodate Ethernet and token ring cards from other vendors. NDIS also permits the use of parallel transport stacks in the PATHWORKS for DOS and PATHWORKS for OS/2 products. Digital also supplies NetWare drivers for its DEC EtherWORKS cards for use on Novell networks.

Macintosh Platforms

The Apple Macintosh PC embodies an integrated hardware and software system architecture that has not been cloned by competitors and thus has fewer variants than the Intel-based PCs. Macintosh PCs use the Motorola 68000 series microprocessor. The later versions of these microprocessors provide 32-bit operations on a 32-bit address bus, with virtual paged memory. Application programmers are largely shielded from the underlying hardware by an extensive operating system application programming interface (API).

All current Macintosh PCs are equipped with bitmapped graphics, sound-generating hardware, a desktop bus for keyboard and mouse connection, and an AppleTalk network communications port. Some Macintosh PCs have system buses that permit peripheral card extensions. All Macintosh PCs allow communication by means of the AppleTalk family of protocols over the LocalTalk LAN.[2] Ethernet/LocalTalk bridges and routers are available from several vendors. Digital's PATHWORKS product family includes VMS AppleTalk transport stacks and an AppleTalk/DECnet gateway.

PC Operating Systems

The PATHWORKS product set supports several client operating systems, namely MS-DOS, Microsoft Windows, Apple Macintosh, and OS/2, a joint effort of IBM and Microsoft.

MS-DOS Operating System

Microsoft's MS-DOS (and IBM's PC-DOS) operating system evolved as a collection of services for a single-tasking, Intel-based PC. In addition to file and print services, DOS provides a simple framework for I/O, memory management, and other system services. A command line interpreter is used to load an application, which may invoke DOS services or take over various hardware functions on its own. Although DOS is evolving in the direction of providing a protected virtual machine environment, applications may bypass or subvert systems services provided by current DOS versions. This complicates the design of DOS client systems services such as PATHWORKS networking software.

Microsoft Windows Environment

Microsoft Windows software operates over the DOS operating system to provide a protected multitasking (nonpreemptive scheduling) virtual machine operating environment and a graphical user interface. Unlike DOS, the Windows environment imposes severe constraints on application structure and interface design, and on the design of system support software such as PATHWORKS network drivers. Although much of the success of the Windows software is due to its ability to multitask traditional DOS applications, there is a rapidly growing number of Windows-specific applications that take advantage of the graphical environment, such as the PATHWORKS eXcursion for Windows server.

Macintosh Client Software

The first Macintosh client was an integrated multitasking hardware and software system with a well-defined application structure and interface definition. Subsequent hardware and software development has refined and extended operating system services. The Macintosh Communications Toolbox, for instance, defines an API that is used by the PATHWORKS Macintosh client to enable Macintosh PCs to participate in a DECnet network.

OS/2 Operating System

OS/2 was conceived by Microsoft and IBM as a protected-mode operating system. OS/2 software features preemptive multitasking, process threads, interprocess communication, and an extensive GUI. OS/2 provides only limited support for DOS applications, partly because of the constraints of the Intel 80286 microprocessor, and has yet to achieve its anticipated popularity. However, OS/2 remains a powerful operating system and applications development environment, and IBM is addressing perceived inadequacies. Digital's PATHWORKS family includes OS/2 LAN Manager server and client offerings.

PC Networks

Even before IBM coined the term PC, microprocessor-based machines were using networks to share expensive hard disks. Sales of networks on which PCs act as both servers and clients have undergone tremendous growth and have outpaced minicomputer networks in the last several years. The most common service offered by PC networks is transparent access to remote files and printers, which permits PC applications to share resources provided by a network server.

The popularity of PC networks has also spawned a variety of distributed applications such as database, electronic mail, and group productivity products. Most PC client-server applications are simply PC applications that simultaneously share files stored on a remote file server. These applications use a file server to achieve their distributed nature.

PC networks are implemented over more than a dozen underlying physical layers; Digital's PATHWORKS products support Ethernet, token ring networks, and asynchronous lines. All minicomputer and mainframe vendors have products that permit PCs to obtain services from their enterprise-wide networks. Digital's PATHWORKS for VMS and PATHWORKS for ULTRIX products provide transparent file and print services to DOS, Windows, OS/2, and Macintosh PC clients. PC files stored on the VMS or ULTRIX operating system may be accessed by other PCs or by users of the host operating system. In addition, PATHWORKS products provide database access, X Window System support, terminal emulation, electronic mail, and many other services familiar to those in a Digital environment.

As noted above, PC networks use many physical networking protocols. In the following sections, we describe PC transport protocols and the application-level service protocols used to encode the remote service primitives.

Transport Protocols

Commercial PC networks use a wide variety of transport and service protocols. Although minicomputer transports are available to meet some needs, most vendors have introduced their own to address concerns such as performance and size, which are critical in competitive concerns such as performance and code size.

The network basic I/O system (NetBIOS) software, developed by IBM, defines an interface to a connection-oriented transport, a connectionless datagram service, and a name service API.[3] In addition to being the Microsoft LAN Manager transport interface, NetBIOS has become a widely accepted standard for PC applications communicating directly with transports.

Figure 1 shows that NetBIOS can be implemented by PC network vendors over a variety of underlying transports. Digital's PATHWORKS products have NetBIOS interfaces to the DECnet protocol and the transmission control protocol/internet protocol (TCP/IP).[4,5] Other popular commercial transports incorporating NetBIOS interfaces are the internet packet exchange (IPX), the Xerox Network System (XNS), and the NetBIOS extended user interface (NetBEUI). Many of these transports also have a native transport API that allows the application to make use of features not available through the NetBIOS interface.

The TCP/IP protocol family is beginning to achieve some visibility in the PC network market. At first largely associated with UNIX and ULTRIX networks and Sun Microsystems' Network File Service (NFS) protocol, TCP/IP has been lately offered as an underlying transport for NetBIOS in several vendors' products, including Digital's PATHWORKS family. In addition to transparent file and print services, PC users of TCP/IP require access to a variety of tools and utilities, such as mail and terminal emulation, which may resemble UNIX or ULTRIX tools and utilities. Digital's PATHWORKS family has adopted the approach of maintaining parallel TCP/IP and DECnet implementations, both of which have a PC-centric rather than a host-centric orientation.

The PATHWORKS TCP/IP implementation operates over either an Ethernet or a token ring network,

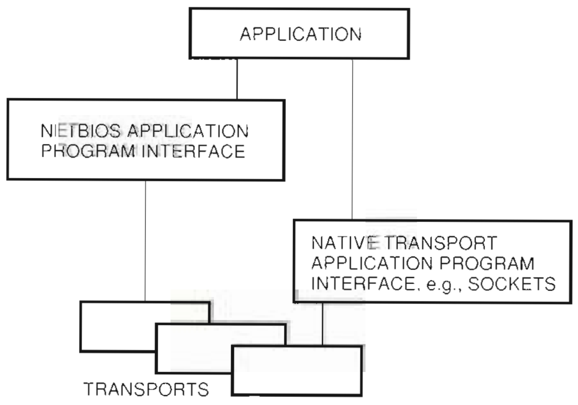

APPLICATION
NETBIOS APPLICATION PROGRAM INTERFACE
NATIVE TRANSPORT APPLICATION PROGRAM INTERFACE, e.g., SOCKETS
TRANSPORTS

Figure 1 NetBIOS and Native Application Program Interfaces

and provides a file transfer protocol (FTP) utility, a TELNET terminal emulator, and a Berkeley Software Distribution (BSD)-like socket interface for application developers.

Many of Digital's customers have extensive DECnet networks. Digital's PATHWORKS product family provides PC clients with full Phase IV end-node functionality, including file access listener (FAL), network file transfer (NFT), command terminal (CTERM), and network utilities. PATHWORKS products also support a NetBIOS implementation that uses the DECnet protocol as a transport. The PATHWORKS DECnet implementation operates over either an Ethernet or a token ring network and provides a BSD-like socket interface for application developers.

NetWare software from Novell Corporation is a popular family of PC network services. The internet packet exchange protocol is Novell's derivative of the Xerox internet datagram protocol. IPX is the network transport that underlies SPX, a sequenced reliable protocol. IPX is also used by the NetWare core protocol, NCP. Novell also supplies an implementation of the NetBIOS interface over the IPX protocol. Digital supports the IPX/SPX protocol on DOS clients through the PATHWORKS for NetWare coexistence product, and has announced plans to integrate NetWare protocols into PATHWORKS products in a way that parallels current use of LAN Manager protocols.

The AppleTalk family of protocols employed by Macintosh PCs accommodates three hardware layers: token ring, Ethernet, and LocalTalk. AppleTalk includes a datagram delivery protocol, routing and name binding protocols, and several session-level and service protocols.

For efficiency, many PC network vendors have invented their own protocols. For example, both the IBM/Microsoft NetBEUI and the 3Com Corporation NBP transport protocols have been optimized to work on LAN topologies.[6] Digital's PATHWORKS software provides the local area transport (LAT) and local area system transport (LAST) protocols on several of its client platforms; these protocols are used to access terminal services and InfoServer disk services.

Service Protocols

Service protocols encode high-level service requests at the application layer; these protocols are often vendor-specific. Typically, an application issues a standard I/O request, such as "open file," to a systems interface to obtain transparent access to a remote file or print service. The request may be either intercepted (e.g., in Novell's NetWare software on DOS) or channeled through the operating system (e.g., in the Microsoft LAN Manager or Apple Macintosh software) to a redirector or shell software module that encodes it into a service protocol packet. The redirector then sends the service request to the local transport. When the response packet arrives from the service provider, the redirector interprets the service protocol and provides the application with the appropriately formatted response. The redirector may also provide an API for access to nontransparent services such as peer-to-peer communication and management of a remote server. Figure 2 illustrates the role of service protocols in fulfilling a client request.

The Microsoft LAN Manager redirector software uses the server message block (SMB) protocol to access remote file and print services.[7] This protocol may run over multiple transports, each transport accessed by means of a NetBIOS interface. The redirector also provides a client API over the SMB protocol for many nontransparent services such as peer-to-peer communications via named pipes, a messaging service, and remote server management.

Novell's NetWare software uses the NCP protocol to access remote file and print services. This popular service protocol runs only on the IPX transport stack. The NetWare shell provides client APIs over NCP for many nontransparent services such as transaction tracking, semaphores, and remote server management.

Apple's AppleShare software uses the AppleTalk suite of protocols. These protocols include the AppleTalk filing protocol (AFP) and printer access protocol (PAP), which permit transparent file and printer redirection.

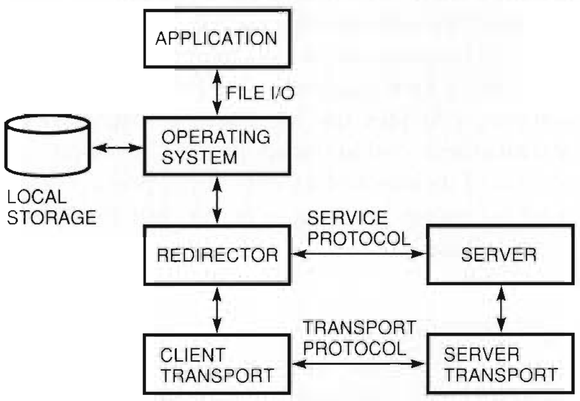


Figure 2 Service Protocols

Sun's NFS system has widespread multivendor support in UNIX and ULTRIX environments. There are a variety of PC products that work over the IP protocol family to provide file services from a standard UNIX or ULTRIX NFS server.

PATHWORKS Product Family

Commensurate with Digital's role as a network integrator, the PATHWORKS product family is large and diverse. In the following sections we characterize the PATHWORKS family by its client platforms, server platforms and services, and physical networks and network protocols. Table 1 shows the history of the PATHWORKS product family.

Since its introduction in 1986, the PATHWORKS product family has continued to expand the list of client platforms, servers, and transports it supports. The most popular client platforms are Intel-based and operate under DOS and/or Microsoft Windows. These clients can be serviced by VMS, ULTRIX, and OS/2 servers. The Macintosh clients can be serviced by VMS servers.

The PATHWORKS product family offers transparent file and print services through two technologies: the Microsoft LAN Manager is used for DOS, OS/2, and Windows client platforms; AppleShare is used for Macintosh platforms. In addition, on DOS and Windows platforms a dual-service stack approach is used to allow these platforms to access native NetWare services through the PATHWORKS for NetWare Coexistence product. Table 2 shows how clients and servers can be connected by means of different transports. The first column is a list of the supported servers; each cell shows the transports that can be used to connect the client and the server.

The Macintosh client also supports the DECnet transport. However, file and print services are only available through the AppleTalk stack. Clients also have access to a number of transport gateways, including AppleTalk-DECnet, X.25, and the System Network Architecture (SNA), the latter two through Digital network products.

The default PATHWORKS network protocol is the DECnet protocol. TCP/IP is available as an optional add-on to the base platform. The DECnet protocol allows the user to access the following services in addition to the transparent file and print services:

- A full set of management tools (e.g., the DECnet network control program for managing the transport).

Table 1 Product History

Area Supported	1986–89	1990	1991
File and Print Service	LAN Manager AppleShare	LAN Manager AppleShare	LAN Manager
Server	VMS	VMS ULTRIX OS/2	VMS ULTRIX OS/2
Transport	DECnet	DECnet AppleTalk TCP/IP	DECnet AppleTalk TCP/IP NetBEUI
Network	Ethernet	Ethernet LocalTalk	Ethernet LocalTalk Token Ring
Clients	DOS	DOS OS/2 Macintosh	DOS Macintosh OS/2 Windows 3.0 NetWare Coexistence

Table 2 Client-Server Transports

	Client Platforms			
Server Supported	DOS	Windows	OS/2	Macintosh
VMS	DECnet TCP/IP LAST	DECnet TCP/IP LAST	DECnet TCP/IP	AppleTalk
ULTRIX	DECnet TCP/IP	DECnet TCP/IP	DECnet TCP/IP	
OS/2	DECnet TCP/IP NetBEUI	DECnet TCP/IP NetBEUI	DECnet TCP/IP NetBEUI	

- The NFT utility for transferring files to systems that do not have server software.
- A remote disk (as opposed to remote file) mechanism over the LAST protocol. This mechanism allows access to Digital's InfoServer products that support networked CD-ROMs, i.e., read-only optical disks.
- DOS and Windows terminal emulators operating over the LAT or CTERM protocols, as well as asynchronous lines. The LAT protocol may also be used to attach a local PC printer to a VMS print queue.
- A DOS-based X Window System server that allows the PC to act as a display device for Motif or DECwindows applications.
- A low-end electronic mail utility that provides a PC front end to the VMS and ULTRIX mail systems.
- Development tools in the form of programming libraries for access to peer-to-peer communication with remote applications.

The TCP/IP protocol allows the user access to the following services in addition to those listed above:

- The FTP utility for file transfer
- The ability to use the base terminal emulator to allow operation over TELNET
- The ability to run the DOS-based X Window System server over TCP/IP as well as over the DECnet protocol

Every Macintosh PC includes software to access basic file and print services over the AFP. The PATHWORKS Macintosh product family provides those services on server platforms, but also provides a set of transport protocols and utilities on the Macintosh client. In particular, PATHWORKS products supply a DECnet stack with file transfer and management utilities, a LAT implementation and terminal emulator, and an X Window System server implementation that operates over the DECnet or an optional TCP/IP stack. The PATHWORKS Macintosh client includes a programming tool for access to remote databases on Digital platforms.

The PATHWORKS OS/2 client provides a LAN Manager redirector and SMB access to basic file and print services over the DECnet protocol or an optional TCP/IP stack, and a collection of tools and utilities similar to those for the PATHWORKS DOS client. Some features, such as an X Window System server, are lacking.

In addition to the applications included in the base PATHWORKS product, the following client applications are available as layered products:

- eXcursion for Windows, a Microsoft Windows/X Window System server application that allows X Window client applications to share the PC display device with native Windows applications
- X.400 mail, which provides PC front-end access to Digital's X.400 mail server products
- Conferencing, which provides a PC front end to VAX Notes
- Videotex, which provides a PC front end to Digital's Videotex servers
- DECquery software, which provides a PC front end to structural query language (SQL) services

Digital also provides development tools for building distributed applications on the PATHWORKS base system. These development tools include database access to a host-based SQL server by means of the SQL services and distributed transaction processing through the DECtp for ACMS product.

Summary

The PATHWORKS product family provides direct access to the local and wide area enterprise environment from desktop devices. Clients can access multiple file and print servers, gateways, database servers, transaction processing systems, and electronic mail systems on a variety of server platforms in a consistent manner from multiple desktop platforms.

The services provided by the PATHWORKS product set are the foundation for the integration of desktop applications with host system services such as those available with the VMS, ULTRIX, and OS/2 systems. PATHWORKS network software makes it possible to develop front-end processors for today's host-based applications and to design new distributed applications. Hence, PATHWORKS products allow the existing computing infrastructure to progressively evolve towards a distributed model.

References

1. B. Baldwin, *Local Area Communication Service,* Metric Note LAN 40 (Stamford, CT: Gartner Group, Inc., December 1991).

2. G. Sidhu et al., *Inside AppleTalk,* 2nd ed. (Reading, MA: Addison-Wesley, 1990).

3. *IBM NetBIOS Application Development Guide* (Armonk, NY: IBM Corporation, Document No. S68X-2270-00, 1987).

4. *Protocol Standard for NetBIOS Service on a TCP/UDP Transport: Concepts and Methods,* Internet Engineering Task Force RFC 1001 (March 1987).

5. *Protocol Standard for NetBIOS Service on a TCP/UDP Transport: Detailed Specification,* Internet Engineering Task Force RFC 1002 (March 1987).

6. *Local Area Network—Technical Reference* (Armonk, NY: IBM Corporation, Document No. SC30-3383-2, November 1988).

7. *X/Open Developer's Specification—Protocols for X/Open PC Interworking: SMB* (Reading, U.K.: X/Open Company Limited, Document No. XO/DEV/91/010, 1991).

Edward W. Bresnahan
Siu Yin Cheng

PATHWORKS for VMS File Server

The PATHWORKS for VMS file server integrates industry-standard personal computers with VAX VMS systems over a communications network. It implements Microsoft's server message block (SMB) core protocol, which provides resource sharing using a client-server model. The server provides transparent network access to VAX VMS FILES-11 files from a PC's native operating system. The architecture supports multiple transports to ensure interoperability among all PCs connected on an open network. Due to the performance constraints of many PC applications, data caching and a variety of other algorithms and heuristics were employed to decrease request response time. The file server also implements a security model to provide VMS security mechanisms to PC users.

Coupled with the PATHWORKS for DOS or PATHWORKS for OS/2 product, PATHWORKS for VMS creates a distributed computing environment, based on a client-server model. This environment allows personal computer (PC) users to access VMS system resources transparently. PC clients access the system server from their native operating systems, typically MS-DOS, as if it were local to the PC. The VAX VMS system resources to be shared, i.e., files or printers, are offered as services over the network to PC clients. The computer systems providing the shared resources are referred to as servers; and the PCs requesting the resources as clients. The SMB protocol from the Microsoft Networks/OpenNET (MS-NET) Architecture was chosen to provide file sharing from a VAX VMS system to MS-DOS and OS/2 clients.[1] The SMB protocol is a command/response application-layer protocol designed to provide file sharing in a PC network. Since SMB is an application-layer protocol, it is transport independent and thus can be implemented over heterogeneous networks.

Central to this environment is the file server, the component that processes the SMB requests to provide file and print sharing along with management functions. The file server maps SMB file requests to the appropriate calls for the VAX VMS FILES-11 file system interface and honors applicable security mechanisms. MS-DOS and VAX VMS systems have different file systems and security models. To integrate these different environments, mapping policies, along with an architecture appropriate for the VMS system, had to be developed and implemented.

This paper describes the design and implementation of a nondedicated personal computer file server (PCFS) on a VAX VMS computer system. It details the PATHWORKS for VMS file system and discusses its transport layer interface and performance considerations, including data caching effects and disk space allocation. The paper then explains file sharing among server processes in a cluster environment and concludes with a discussion of the server configuration and management interface.

File Server Architecture

The file server is implemented as a single, multithreaded, nonblocking detached process with an associated permanent DECnet object. This user-mode process is privileged and has a high priority. Figure 1 shows the architecture of the server. Only one file server process exists on any one computer to handle all client requests. An alternative choice would be to have multiple processes service the clients. The use of a single process reduces system resource requirements and eliminates the latency that is incurred from context switches among the multiple server processes. Also eliminated is the latency that results from process creation at the time a client connects.

A threads package with multiple independent threads of execution within a single process supports multiple clients and periodic operations within the file server. The file server creates a thread for a client when it requests establishment of a virtual circuit to the file server. The thread is

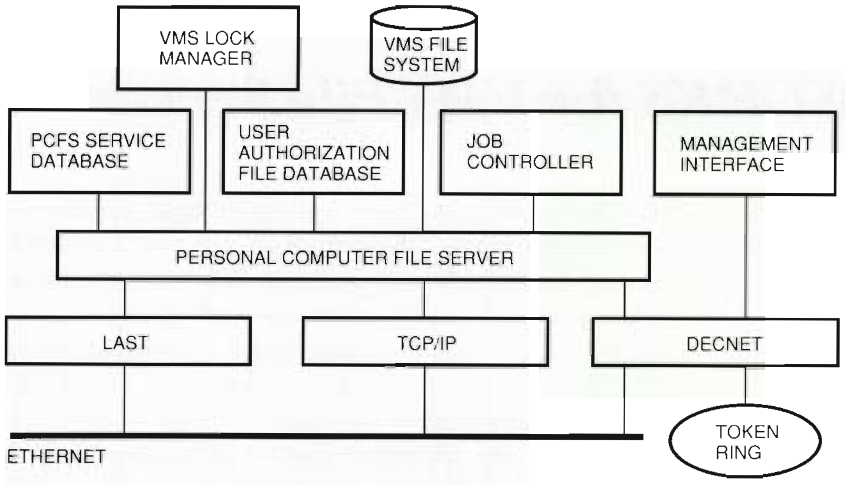


Figure 1 Server Architecture

deleted when the client terminates its connections. A client's thread carries out the operation specified in the request SMB without blocking the process. With this scheme, processing SMB requests is sychronous with respect to the client, yet asynchronous with respect to the file server process.

Since a server process may be processing the requests of hundreds of clients simultaneously, the server operates in real-time. The threads package contributes to these goals by providing an environment in which the process never enters a wait state and a client thread is safe from CPU starvation. Preventing the process from blocking is accomplished by performing all file I/O asynchronously and by calling operating system routines asynchronously when possible. Starvation is prevented by scheduling clients using a nonpreemptive first-in, first-out (FIFO) scheduling algorithm. With this policy, a thread executes until it voluntarily yields, usually due to an I/O operation or an operating system call. Using a nonpreemptive scheduling algorithm also eliminates the latency that would result from a thread switch in a preemptive environment.

PATHWORKS File System

A file server needs to provide transparent file access to a VMS file system and ensure file accessibility between DOS and VMS users. Since these operating systems have different file systems, PATHWORKS for VMS must store the files in VAX VMS FILES-11 format and provide a mapping algorithm to bridge the two operating systems. Because the OS/2 and DOS systems use the same file system, the mappings performed to address the difference between the DOS and VMS systems can be applied to support transparent file access from an OS/2 client.

File Name Mapping

DOS and VMS FILES-11 support different naming syntaxes. DOS supports 8.3 naming format; that is, the file name is composed of a maximum of eight characters with a maximum of three characters as the extension. In contrast, the VMS FILES-11 file name supports 39.39 format and includes a third component, the file generation number. In addition, the legal character set for a file name is larger in DOS than it is in the VMS system.

The PATHWORKS file server does not include a mapping algorithm to convert a 39.39 VMS file naming syntax to be accessible to DOS. Any VMS file that DOS system users need to share must be created with a file name that conforms to DOS 8.3 format. Since the 8.3 naming format maps directly to the 39.39 format, no mapping algorithm is required to guarantee a VMS system user access to files named by a DOS system user.

To overcome the difference in character sets, a comprehensive mapping algorithm was written to ensure shareability and transparency. Since neither operating system is case sensitive, the file server changes the file name to uppercase before any operation is performed on the file. The legal character set for VMS FILES-11 file names includes uppercase alphanumerics, dollar sign, hyphen, and underscore. The character set in DOS includes all noncontrol characters with the exception of a few special signs. The PATHWORKS server maps the character sets based on the following rules:

- All alphanumeric characters are changed to uppercase letters; any character that is valid in a VMS file name is passed through unchanged.
- All other characters are changed to two underscores, followed by two hexadecimal digits that represent the ASCII code of the character being mapped.

VMS FILES-11 allows multiple versions of a file to be generated and stored in a directory. These files are identified by the numeric component, which represents the version number, of a file name. There is no equivalent concept in the DOS system. The PATHWORKS server maps the highest version (or most recent generation) to be accessible to DOS. Similarly, the server, when creating a file on behalf of a DOS client, generates the file with a version limit of 1. To preserve and honor the version limit information for the VMS environment, the server preserves the VMS file attributes of previous versions of the file. Consequently, if the file is created by a VMS user, and is later updated by a DOS user, a new version of the file is generated, and the version limit information is preserved.

Directory Mapping

The VMS system requires a directory name to end with "dir" as an extension, but the DOS system does not post any restriction in this area. PATHWORKS maps directory names in DOS by including the ".ext" characters as part of a directory name. Since the period is not a legal character for a DOS directory, it is mapped using the double underscore followed by the hexadecimal digit rule. Any directory name in DOS that conforms to the VMS directory naming syntax is passed through untouched.

DOS File Attribute Mapping

Both file systems associate a set of attributes to the files, but the file attributes on a DOS file do not have a one-to-one correspondence with those on a VMS file. A DOS file has four types of file attributes: archive, system, hidden, and read-only. The concepts of archive, system, and hidden are not recognized in the VMS file system. PATHWORKS software stores the DOS file attributes in an application access control entry when creating a file on behalf of a PC workstation. Furthermore, the read-only attribute of a DOS file is mapped to the read-only bit of the record management services (RMS) protection field for system, owner, and group.

File Organization

A DOS file is organized as a byte stream, but a VMS file is organized as collections of records. Although the VMS system supports a form of stream file, most VMS files are stored in record format. Furthermore, a VMS file with a stream record format does not map directly to a DOS stream format. This poses an interesting problem in integrating VMS and DOS file systems.

Since PATHWORKS software provides transparent access to the VMS host system, a DOS client views all files on file services as streams of bytes, just as if these files were stored locally. When the server creates a file on behalf of a PC, it specifies the file organization as sequential with stream record format. Thus, the byte stream characteristic of the DOS system is preserved.

The more complex part of the problem is to resolve the shareability issues between VMS and DOS applications. The PATHWORKS server is implemented to provide the necessary conversion between VMS and DOS file organization on stream files. The file server views a file as stream if it can read and write the file without regard to any record boundaries. This includes any files with file organization as sequential and record format as stream, stream_cr, stream_lf, and undefined, as well as fixed. If a sequential file has fixed record format, it must conform to record size and attributes as follows: even with no record attribute; 512 with no block_span; and power of 2 with no block_span. Thus, an RMS overhead in reading and writing these files is avoided.

Any file that does not meet the criteria of the stream category is said to be nonstream. The PATHWORKS server provides read-only access to any VMS nonstream file. This is achieved by using a VAX C run-time library call that provides stream file semantics and a conversion algorithm to properly map any carriage return and line feed information. The file server cannot support writing to these files because the SMB protocol does not preserve record boundary information. Thus, the protocol makes it impossible for the file server to guarantee data integrity when updating a nonstream file.

Byte Range Locking

The MS-NET architecture allows for concurrent access to server-based files by multiple clients. PC applications acquire this functionality through the MS-DOS byte range locking calls. These calls allow PC applications to lock and unlock ranges of bytes in a file and to detect conflicts. Conflicts occur when part or all of a range specified to be locked has been locked from a previous call. In contrast, the approach taken by RMS provides locking on a record basis. RMS uses the VMS distributed lock manager to implement this functionality. Unfortunately, the lock manager is not well suited to implementing byte range locks because the byte range is represented in a form that allows the lock manager to arbitrate access. Therefore, the file server implements its own lock database and arbitrates access to shared files. Internally, the server process maintains a list of locks for each file the server has open and arbitrates access based on these lock structures. Files opened by the file server cannot be shared with other VMS processes because the file server has an exclusive mode lock on each file it has open through the VMS lock manager. The exclusive mode lock guarantees protection from other VMS processes.

Open Mode Mapping

The DOS file system defines open access modes to allow applications to synchronize shared access to a file. The open modes are deny_none, deny_read, deny_write, deny_read_write, and compatibility. Each provides a different level of file sharing capability. Although these modes do not map directly to the VMS file system, no mapping is needed to handle the differences.

The PATHWORKS server opens a file that is being accessed by a client with exclusive access on the VMS system. It assumes the responsibility to arbitrate shared access among multiple clients. The server supports DOS open access modes by implementing the shared access resolution algorithm described in the SMB protocol specification.

PATHWORKS Transport Layer Interface

The PATHWORKS for VMS product supports multiple transports through a common transport layer interface. These include the local area system transport (LAST), the transmission control protocol/internet protocol (TCP/IP), and the DECnet transport protocol over Ethernet and token ring networks. This well-defined, uniform mechanism dynamically adds support for network transports and protocols. By conforming to this specification, transports can be added to a server platform without upgrading or changing the existing file server.

The performance goals of the file server had an impact on the development of the transport layer interface. The file server utilizes an optimized transport layer interface that reduces buffer copies and eliminates some of the standard VMS I/O paths. This optimized interface is used with the LAST transport and is described in detail in "The Development of an Optimized PATHWORKS Transport Interface" paper in this issue.[2]

Performance Considerations

Achieving an acceptable level of performance from a nondedicated file server layered on a general-purpose operating system proved to be a challenging task. One of the performance goals for the file server was that it perform tasks within 10 to 20 percent of the speed of a dedicated PC file server running on a similarly sized CPU performing the same tasks. This goal was achieved by employing a variety of caches, algorithms, and heuristics. Many of these heuristics were based on the analysis of the SMB messages passed between the server and the client for typical PC applications. As discussed in this section, the response time of the server is improved if the memory contains the information necessary to satisfy a request when it arrives.

Data Caching

An obvious approach to implementing the read and write functions in the file server is to issue these operations to the FILES-11 file system, wait for their completion, and then send a response to the client. This method is simple and persistent, but does not perform well due to the bottleneck formed at the FILES-11 interface and disk. The file server implements a software write-behind data cache to reduce this bottleneck and to eliminate waiting for disk writes to complete before returning a response to the client. Caching is a technique used to decrease access time to information by using a faster intermediate medium to store the most commonly accessed pieces of information. The caching algorithm implemented by the server is a logical block cache. The cache is a region of memory that is segmented into fixed-sized buffers. Each file opened by the server has a dynamic set of buffers that increase and decrease based on a least recently used (LRU) algorithm.

Effects on Client Read Requests Although this is an optimal environment for servicing read requests, reserving data in memory to satisfy all read requests is not practical. A number of mechanisms were implemented to approach the ideal. The data cache retains recently accessed data in memory with the expectation that it will be referenced again soon. This is based on the concept of locality of reference, both spatial and temporal. Once the server receives a read request, it determines if the buffers associated with the read request are in the cache by using a hashing algorithm for the lookup function. If the data to satisfy the read request is in memory, it is immediately returned to the client, and the file system access is eliminated. If some of the data needed to satisfy the request is not in the cache, then reads are started on each of the cache buffers needed to satisfy the request. Once all data is read into cache memory, a response is formed and returned to the client.

Effects on Client Write Requests When the server receives a client write request, three processes are performed. The cache buffers needed for the specified write range are located, the client data is copied to the cache buffers, and a response is sent to the client. The data copied to the cache is written to the disk at a later time. This write-behind scheme allows write requests to be serviced quickly because the response is returned to the client before the write to disk completes. By not synchronizing on-disk write completions before returning a response, the turnaround time of client write requests is greatly reduced. The cache is also optimized when a client write request is received and a disk read operation is in progress for the range. In this case, the data being written to the cache is copied into an intermediate buffer and merged with the data from disk after the read operation completes. These intermediate buffers are known as ghost buffers, since they are not visible from the buffer hash table.

Writing Data to Disk Since the file server acknowledges write requests before performing the write operation, a mechanism is needed to write the cache buffers to disk and ensure data integrity. The file server implements a permanent thread, the flush thread, dedicated to this task. The flush thread starts disk write operations on buffers that contain modified data. Flushing data to disk occurs (1) periodically, based on a user-configurable interval; (2) when a file is closed; (3) when the ratio of dirty to free cache buffers reaches a user-configurable threshold; and (4) when cache buffers are not available to support the current request.

On the VMS system, RMS also employs a write-behind algorithm similar to the one used by the file server. RMS is not used by the file server for disk reads and disk writes for performance reasons. The crossing of the VMS architectural boundary that occurs during RMS calls adds an unacceptable amount of processing time to the read and write paths. The file server uses the VMS queued I/O (QIO)/extended QIO processor (XQP) interface, which is below the RMS layer, to read and write data to disk.

Disk Space Allocation

Sufficient disk space must be available for any write operation that is performed as a background operation. To allow sufficient space, any disk allocation must be completed when the write request is received. This restriction slows down write operations which, in turn, results in file expansion. Performance testing in this area shows that such expansion operations can reduce the server's response time in the overall operating environment. To alleviate this problem, the PATHWORKS server preallocates a fixed amount of disk space, often much greater than required, to complete the current write request, in anticipation of further file expansion. This mechanism greatly reduces the system overhead incurred in disk allocation; thus it improves the overall response time to write operations.

Read Ahead

Another mechanism used by the file server to improve the turnaround time of read requests is read ahead. As with data caching, the goal is to increase the probability that data referenced in the near future will be in the cache. Read ahead is the process of prefetching previously unreferenced data from the disk into the cache. Data is prefetched into cache memory under several conditions. When a file is opened, the first two cache buffers of the data are read from the disk into the cache. Data is also prefetched when the server detects that the file is being accessed sequentially. The SMB protocol also supports read ahead. The protocol provides a field in the read request that specifies the amount of data that the client intends to read in the future. This advisory field is used by the server to initiate prefetches.

Directory Search-ahead Cache

A DOS directory operation can translate to multiple exchanges of request and response operations between the server and client. This behavior is inherent to the SMB protocol definition. The file server initiates a search-ahead thread when the first request is received. While the PC is processing the first response, the search-ahead thread accumulates directory information in a circular buffer. Thus, this information is available in memory for subsequent requests.

Open-file Cache

Operations, such as create, open, and close, impact performance in the VMS system. Benchmark tests show that these operations become blocking factors for a fast performance server. This problem is compounded by the inherent behavior of many PC applications because they often use the result of an open operation as a deterministic tool on file accessibility. Frequently, files are opened and closed and reopened in consecutive requests. To minimize the overhead incurred for these operations, the PATHWORKS server implements a cache to store opened file information. This open-file cache maintains the file header information after the file has been closed by the user for a short duration. If a user requests to open a file that is already cached, no request to VMS FILES-11 system is required. This greatly reduces the response time of the server on the second open request.

Furthermore, many DOS database applications use index files to synchronize data access. These files are frequently accessed by many DOS users when working in an networked office environment. Open-file caching is beneficial to this environment because it incurs a minimal amount of open requests to the VMS file system.

Byte Range Locking Back-off Algorithm

The file server implements an algorithm to improve overall performance of the server and network when PC applications are sharing files and using byte range locking to arbitrate access. The analysis of many networked PC database applications revealed that a client typically entered a tight retry loop when it detected a lock conflict. This spinning produces an excessive amount of lock-related network traffic, especially for very fast clients. The server also has to spend a significant amount of time processing these numerous lock requests. The server attempts to regulate this lock traffic and reduce its lock processing time by deferring the return of the response when a lock conflict is detected. If a request to lock a range conflicts with a previous lock, the server makes repeated attempts to access the range using a pseudorandom exponential back-off algorithm to determine the retry interval. If the lock conflict is not resolved after a user-configurable time period, the server returns a response indicating a lock conflict. By deferring this response to the client, the server exercises flow control over clients spinning on locked regions of the file. The implementation of the pseudorandom exponential back-off algorithm prevents the server from using an excessive amount of CPU time to determine if the locked byte range has been unlocked.

Security

The VMS operating system offers a well-defined security architecture, but DOS has no comparable security scheme. Since the PATHWORKS file server is implemented as a privileged process, it is necessary to control file access on the VMS host system from a DOS client. There is no one-to-one correspondence between a DOS user and a VMS user. That is, in the PATHWORKS environment, each network client, much like a terminal in this respect, can be multiple VMS users. The problem is to ensure maximum shareability among PC clients and maintain the desired level of VMS security.

The PATHWORKS file server implements two types of securities: share and user. It makes use of the PCFS$SERVICE_DATABASE to control access to a share area; and the VMS user authorization file (UAF) database to control access to directories and files based on a VMS user account. A share, referred to as file service, is a VMS directory that can be accessed by PATHWORKS clients. PATHWORKS software defines three types of file services: system/application, common, and personal. Access to file services is based on VMS user account information. A privileged system manager must explicitly grant user access to system/application and common services. The system manager must also specify the types of access: read, write, or create. This information is stored in the PCFS$SERVICE_DATABASE. Access to personal service is implicit with the existence of a user account.

To provide maximum shareability among PC clients, PATHWORKS software includes a default user account. When accessing a file service that has been granted to the default account, each PC assumes the identity of the default account. Thus the access, though it might be issued by different PC users, is viewed as the same user. This mechanism provides a "share level" of security.

A more restrictive environment is achieved by providing access to a share area based on individual user account. When a PC client establishes access to a service, it presents a user account and its corresponding password. This information is authenticated based on information returned by the sys$getuai system service call. The PATHWORKS server then verifies that this user has been granted access to the service.

Access to a file service does not necessarily imply access to any individual files. In order to preserve the desired level of VMS security, PATHWORKS honors access control entries. The server ensures access to a share area as defined in the database by mapping the access types to two identifiers: pcfs$read and pcfs$update. These identifiers are added to the root directory of a share area, and to any files that are created, when appropriate. As the server impersonates the user, the appropriate identifier is associated when access privilege to files and directory is checked. This security implementation is not applicable when servicing a personal area. Access to files stored in a personal area is based on RMS protections mask.

To ease system management tasks, PATHWORKS software implements "group" support. A group is a collection of users. A PATHWORKS group has no dependency on user group identification code. When a share is granted to a group, each member of the group gains access. Note that authentication is still performed based on an individual user account.

Since a DOS client can gain access to the VMS environment, it is imperative that the file server support the VMS system's break-in evasion mechanism. The server honors the login-related system parameters. These parameters are read at the file server start-up, and the values are in effect for the duration of the server process. The server tallies any failed or unsuccessful login attempts. When the file server receives a connection (login) request to service, the file server extracts the related counter information from the UAF and adds it to its internal counter to determine whether evasive action is to take place. When a break-in is detected, the server takes the appropriate evasive action and signals the condition in the server log file.

Printing Support

The server process also implements the printing functionality specified in the SMB protocol. The file server implements the print-related commands by using $SNDJBC and $GETQUI system services to communicate with the VMS job controller. Each print service available to clients has a VMS print queue associated with it.

The VMS system has a much richer printing environment than the one provided to the PC clients through the SMB protocol. The PATHWORKS server provides VMS printing features to the clients by extending the SMB protocol to accommodate PATHWORKS needs. These protocol extensions are described in the section Digital Protocol Extensions.

File Sharing among Server Processes

Each node on a VAXcluster system can be a host for the PATHWORKS server process. One of the more challenging problems in supporting VAXcluster systems is the synchronization of file access by multiple server processes. As stated earlier, the PATHWORKS file server requires exclusive access to files that are opened by PCs in order to support byte range locking in DOS. Furthermore, in a cluster, each server process needs the ability to provide identical access to the same resources.

PATHWORKS software implements its own lock management algorithm to resolve file access conflicts in a VAXcluster system. Although multiple server processes are allowed in the environment, only one process can handle the requests to a file that is accessed by PC clients. By using the VMS lock manager, the server process that services the first open request acquires an exclusive mode lock on the file. It thus becomes the master of the file and is responsible for synchronizing access requests to the file. When a server process is requested to service a file that has another PATHWORKS server as its master, it makes a network connection to the master process and forwards the requests. This process serves as the routing agent. It communicates both requests and responses between the master server process and the PC client. The master releases ownership when no outstanding open file handles are on the file. File mastering is established on a per file basis.

The rerouting mechanism uses the DECnet transport because its existence on the remote server host is guaranteed in a cluster environment. To minimize the number of required DECnet sessions, the routing agent funnels all forwarding SMBs through an existing session. The forwarding packets include information that the master process can use to differentiate among the clients' access requests.

PATHWORKS Server Configuration

The multithreaded PATHWORKS file server can be considered a small operating system in which each PC is a process (or a thread). In addition to the basic resource requirement that the server be activated, the server requires a set of process resources to support each client thread. These resources can be mapped to VMS process parameters which, in turn, translate into system parameters.

The amount of VMS system resources which the file server consumes is a function of the number of clients and the workload generated by the individual PC. Mapping the PC resource requirement to the appropriate VMS process and system parameters proves to be a complex problem. Since the PC workload profile is unknown at the time of server initialization, the amount of required system resources for the server process can only be estimated.

PATHWORKS system managers include users with little VMS system management experience. The level of VMS system expertise required to configure (or set up) a PATHWORKS server is minimized by the addition of a "configurator." This part of the management functionality is implemented to generate information on required system and process resources when the desired configuration is supplied. During the server start-up phase, the configurator checks for availability of necessary resources and provides appropriate run-time parameters for the launching of the server process.

Management Interface

To provide integration between different file systems, the file server utilizes PATHWORKS specific databases (such as the service database), standard VMS databases (such as the UAF and DECnet databases), and VMS security mechanisms. These entities must work in harmony and be consistent with each other to provide the desired integration. The PCSA_MANAGER utility was designed to manage this environment. It allows users to perform all management tasks related to PATHWORKS software through one utility from a menu-driven user interface or a command line interface. The PCSA_MANAGER utility allows system administrators to manage the following objects: users, services, print queues, logical user groups, the event logger, and the server process. The file server uses interfaces supported by VMS to manipulate VMS specific databases, private interfaces to access PATHWORKS specific databases, and SMB protocol extensions to interact with a server process.

Digital Protocol Extensions

Management of a running server requires a method to send and receive well-defined messages between the server and other processes. The PCSA_MANAGER utility sends a management request to the server; the server processes it, and sends an appropriate response back to the PCSA_MANAGER. The communication channel used for server management is a DECnet logical link. The PCSA_MANAGER issues a connection request to the DECnet object associated with the file server process. The file server receives this request and creates a virtual circuit with a corresponding thread to process requests for this management session. This is similar to a client session.

Since the SMB protocol does not provide commands sufficient to manage a PATHWORKS server, a Digital proprietary protocol was developed to provide this functionality. This protocol is merely an extension of the SMB core protocol; that is, the messages developed for server management have valid SMB headers with command codes that are meaningful only to a PATHWORKS server. This implementation allows remote management of the file server. To manage a server, a management utility only has to establish a virtual circuit and exchange these extended SMBs. Protocol extensions are also used to integrate the VMS print system with PATHWORKS clients, along with other PATHWORKS specific utilities.

Event Logging

The PATHWORKS server includes an event logging mechanism to provide an error and event reporting facility to assist system management. Events are categorized based on server operations, including errors, protocols, security, management, and file-related functions (open/close, read/write). The server uses an event code to determine whether a given event is to be recorded. A Digital extended SMB command toggles these event codes dynamically. The event messages are logged to the file server log file. The overhead is minimized by caching the event messages in a data buffer, which is periodically written out to the log file. A thread is created at server start-up to handle the log file update function. The scheduling of this thread is based on a time interval, with a default value of 60 seconds.

Summary

The PATHWORKS for VMS file server integrates the DOS, OS/2, and VMS operating system environments on a network. The server architecture achieves transparent integration of PCs connected on an open network over multiple transports. Data caching, algorithms, and heuristics were used to increase performance. The PATHWORKS for VMS file server provides PC users with access to the VMS system's resources and security environment.

Acknowledgments

We thank the people, past and present, who contributed to the design and development of the PATHWORKS for VMS file server. We specifically acknowledge Robert Praetorius for his contribution in the design and implementation of the cache component, Phil Wells for his design and implementation of the network interface and transport support, and Jon Campbell for his design and implementation of the network interface. We also acknowledge Frank Caccavale for his work on performance analysis, Alan Abrahams for his direction as architect, and Mark Olson for his leadership of the PATHWORKS for VMS project.

References

1. *X/Open Developer's Specification—Protocols for X/Open PC Interworking: SMB* (Reading, U.K.: X/Open Company Limited, Document No. XO/DEV/91/010, 1991).

2. P. J. Wells, "The Development of an Optimized PATHWORKS Transport Interface," *Digital Technical Journal*, vol. 4, no. 1 (Winter 1992, this issue): 24-30.

Philip J. Wells |

The Development of an Optimized PATHWORKS Transport Interface

Digital's Personal Computing Systems Group developed an optimized transport interface to improve the performance of the PATHWORKS for VMS version 4.0 server. The development process involved selecting a transport protocol, designing appropriate interface test scenarios, and measuring server performance for each transport interface model. The engineering team then implemented the optimized design in the server and performed benchmark testing for specified server workloads. Using an optimized transport interface improved server performance by decreasing the time required to complete the test while maintaining or decreasing the percent CPU utilization.

The PATHWORKS family of network integration software products includes file servers that provide file and print services to personal computers in local area networks (LANs). Developed by the Personal Computing Systems Group (PCSG), the PATHWORKS for VMS version 4.0 server supports the Microsoft LAN Manager network operating system. This server allows PC clients transparent access to remote VMS files. With each new release of the PATHWORKS for VMS product, the PCSG engineering team improved server performance and thus accommodated an increasing number of time-critical PC applications. In version 2.0, we introduced disk services as an alternative to file services for read-only files. We included data caching in version 3.0 of our file server.

For version 4.0, our goal was to increase file server performance by optimizing the transport interface and the data buffering algorithm. To achieve this goal, we evaluated several transport interface designs and measured server performance for various server workloads. We started with the premise that using the standard buffered interface results in increased overhead for each transaction and thus decreases overall CPU availability. Figure 1 illustrates this interface design. The server copies a user data buffer in process context across the kernel service interface to a system buffer in system context, before transferring the data to the network layer.

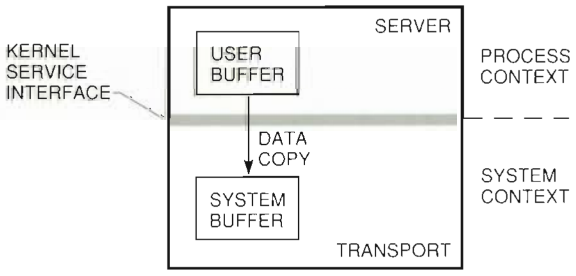


Figure 1 Data Copy with a Buffered I/O Interface

Prior analysis of PATHWORKS server performance over the DECnet transport protocol revealed that when the file server request sizes were large, i.e., 4 to 8 kilobytes (KB), file server performance met or exceeded the performance of other vendors' transports. However, when the transfer sizes were small, i.e., less than 256 bytes, file server performance degraded significantly. Also with small request sizes, our server did not ramp well when many clients were supported in this environment. As illustrated in Figure 2, incremental increases in server workload cause dramatic increases in CPU utilization once a certain workload is reached, i.e., at the knees of the curves, denoted by points A and B. We wanted our server performance to approach that represented by the curve containing point B.

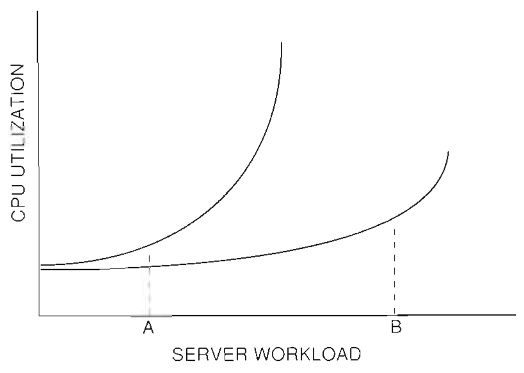


Figure 2 CPU Utilization as a Function of Server Workload

In this way, we could support more clients at the same or less CPU utilization.

Server Performance Analysis

We based our analysis of PATHWORKS server performance on two initial hypotheses:

- The CPU overhead associated with a buffered interface significantly degrades the performance of the server.
- The variable transaction response times inherent in using the standard queued I/O (QIO) interface results in inefficient server performance.

Protocol Selection

To begin our performance analysis, we needed to choose a transport protocol. We considered the DECnet and the local area system transport (LAST) protocols and selected the LAST protocol for the following reasons:

- An advanced development effort on the DOS client software showed that file and print services over the LAST protocol decrease the client memory usage by one-third.
- The PATHWORKS engineering team maintains the LAST protocol and thus, can make any required modifications.
- The VMS operating system implementation of the LAST transport protocol is called LASTDRIVER. LASTDRIVER serves our purpose because it presents a buffering model that permits the passing of multiple noncontiguous data buffers as a single, logically contiguous buffer. Figure 3 shows two physical data buffers, of sizes N and M, being passed to LASTDRIVER as a single message. The second buffer descriptor contains a zero in the next buffer descriptor pointer word. This value indicates the end of the data stream.

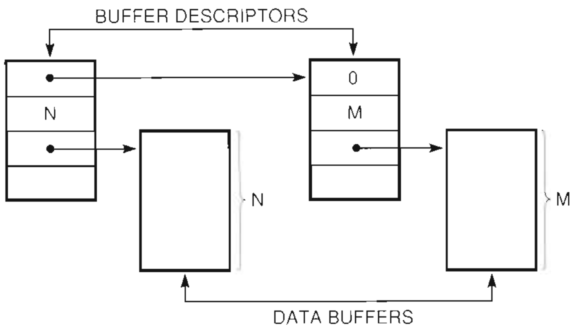


Figure 3 LASTDRIVER Buffering Model

Test Scenarios

After selecting the LAST transport protocol, we created four test scenarios to measure server performance. The first scenario, the kernel model, required developing a VMS device driver that was layered on top of LASTDRIVER. In this model, when the driver receives request data, the data is immediately transmitted back to the client. The driver does not copy buffers and does not schedule a process. This model represents the optimum in performance, because absolutely no work is performed in relation to the request.

The second test scenario required that we develop a user-mode test program. This model performs similarly to the kernel model in that it loops receive data directly back to the client without performing any copy operations. This model differs from the first model in that the driver schedules a VMS process to loop the data back to the client. We then developed the following variations on this test scenario to accommodate three transport interfaces to the VMS process. The second and third scenarios represent optimized transport interfaces with regards to two aspects of a request: the initialization and the completion.

- A standard VMS QIO interface model. This model uses the standard interface provided with the VMS operating system.
- A model that incorporates the standard VMS QIO interface with a process wake-up completion notification. This QIO/WAKE model uses the standard QIO interface to initiate a transport request.

However, the transport queues I/O completion notification directly to the receiving process by means of a shared queue and a process wake-up request. The purpose of this optimization was to avoid the standard postprocessing routines of the VMS operating system.

- A model that includes kernel mode initialization and wake-up completion notification. This CMKRNL/WAKE model uses the transport completion technique of the previously described model. However, we created an entry point into the driver for the test program to call, thereby initiating transport requests. The test program uses the change-mode-to-kernel (CMKRNL) system service to call the driver entry point. This optimization was made to avoid the standard QIO interfaces.

To support the optimized transport interfaces, the test program allocates a buffer in process context and divides it into two sections: the first contains shared queues for moving data between process context and system context; the second contains the test program's shared data buffers. The driver issues a call to the system to double map the shared buffer into system context. Figure 4 shows this double-mapped buffer. Since the buffer is contiguous, the difference between the start of the shared data region in process context and the start of the shared region in system context is a constant, and is used as an offset. The test program accesses the shared region by using a process virtual address (PVA); device drivers access the region by adding the offset to the PVA to compute a system virtual address (SVA), as shown in Figure 5. To accomplish completion notification, the driver inserts the data into the shared queue and issues a process wake-up request for the test program.

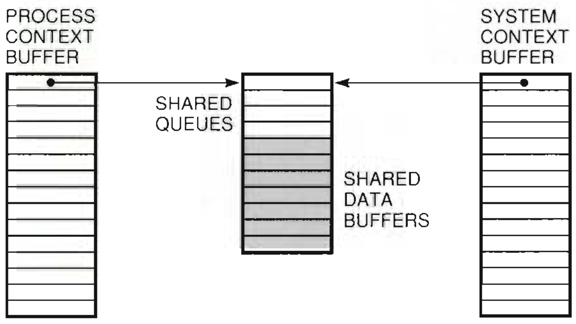


Figure 4 Double-mapped Buffer

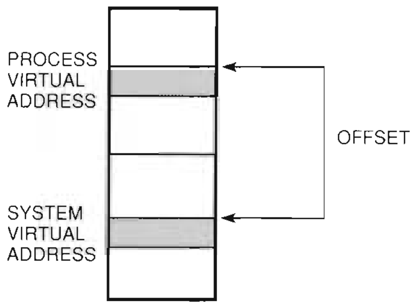


Figure 5 Virtual Address Space

Performance Measurements

Our hardware platform was a VAXstation 3100 workstation. We measured server performance as the difference between the request arrival time and the response departure time, as observed on the Ethernet. Times were measured in milliseconds using a Network General Sniffer. Table 1 presents the test results.

As Table 1 shows, we decreased server response time by using an optimized transport interface. The kernel model yields the best possible performance results. As we move from the standard VMS QIO interface to more optimized interfaces, there is a decrease in transaction response time which represents improved server performance.

Data collected during initial performance testing supported our decision to optimize the transport interface. Occasionally while testing the interfaces, server throughput dropped dramatically, i.e., 30 to 50 percent, for a short time interval, i.e., one to two seconds, and then resumed at its prior rate. Initially, we thought there was a problem with our code. However, the anomaly persisted throughout the development period, so we decided to investigate the cause of the dip in performance.

The VAXstation 3100 system that we used to perform the testing had a graphics controller card installed, but did not include the graphics monitor.

Table 1 Server Performance over Various Interfaces

Interface	Server Performance (milliseconds)
Kernel Model	0.8
Standard VMS QIO Model	2.2
QIO/WAKE Model	1.7
CMKRNL/WAKE Model	1.6

Since the system included a graphics card, the DECwindows login process frequently tried to display the initial DECwindows login screen. This attempt failed because there was no monitor. Therefore, the process was deleted and restarted a few minutes later. We concluded that the temporary drop in server performance we had observed was the effect of the DECwindows start-up process.

The significance of this observation became apparent when we optimized the transport interface, and the effect of this background process activity decreased to less than 10 percent. We concluded that the optimized interface was less susceptible to concurrent I/O than was the standard QIO interface.

Implementation

Once the initial testing of prototypes was complete, we decided to implement the double-mapped buffering algorithm with shared queues. The VAX architecture provides inherent queuing instructions that allow the sharing of data across dissimilar address spaces. It accomplishes this by storing the offset to the data, rather than the address of the data, in the queue header. This technique permits us to insert a system virtual address into a queue in system context and later remove the address in process context as a process virtual address. A second function that these instructions perform is to interlock the queue structure while modifying it. This procedure precludes concurrent access by other code and thus allows the interface to support symmetrical multiprocessing.

We modified the file server to support this new optimized transport interface. To ease the implementation, the QIO interface emulates the DECnet interface in all aspects except one. Since the client-server model is essentially a request/response model, we developed a transmit/receive (transceive) operation that allows the server to issue read buffer and write buffer requests at the same time. This variation reduces the number of system boundary crossings. When the server transmits buffers, these buffers return to the server process by way of a transmit complete queue. When the server receives a new request message, the associated buffer is transferred to the server process via a receive complete queue. To facilitate a transceive operation, we defined a work element data structure. As shown in Figure 6, a work element permits the passing of two distinct data streams: one for transmit and one for receive.

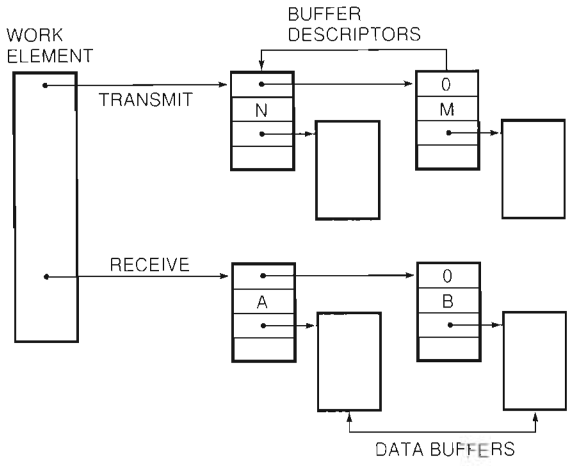


Figure 6 Work Element Data Structure for a Transceive Operation

As development of the client and server software modules continued, we encountered some interesting problems. The following three sections describe several of these problems and how we addressed them.

Microsoft LAN Manager Redirector I/O Behavior

When the Microsoft LAN Manager redirector, i.e., the DOS client protocol equivalent of the VMS file server, generates a read request, it first writes the request for service to the network. The redirector then issues a read request and uses a short buffer to receive only the protocol header of the response message. After verifying that the response was successful, the redirector issues a second read request to receive the data associated with the response message.

This behavior requires lower protocol layers to buffer the response data until the redirector issues a read request to receive the data. In order to buffer the response data for the client, the transport layer needs to allocate an 8KB buffer. An alternative approach to maintaining a dedicated transport buffer is to use the inherent buffering capacity of the Ethernet data link software and the Ethernet controller card, which maintain a cache of receive buffers. This technique requires the transport layer to retain data link receive buffers while the redirector verifies the response message protocol header and posts the actual receive buffer. Once the redirector issues the second read request, the remaining data is copied and the Ethernet buffers are released.

One problem with this approach is that each vendor's Ethernet card has different buffering capacities. In some cases, the capacity is less than the size of the maximum read request. To support such inadequate buffering capability, we inserted a buffer management protocol (BMP) layer between the file server and the redirector. The resulting process is as follows:

The client module communicates its data link buffering capacity to the server module in the session connect message. When the application generates data requests, the DOS redirector packages a server message block (SMB) protocol message and passes it to the BMP layer. This layer adds a small buffer management header to the message and pass it to the transport layer to transmit to the server.

To complete the operation, the file server processes the request, formats an SMB response message, and passes it to the BMP layer. At this interface, the size of the response message is indicated by the transmit buffer descriptors, and a protocol header that describes the response packet is created. If the response message is larger than the client's data link buffering capacity, the driver software segments the response packet into smaller messages and passes these messages to the server transport to transmit to the client. The client module copies the header to the redirector's short buffer and completes the redirector's read request. The BMP layer then waits for the second read to copy the remaining data to the redirector's buffer and releases the data link buffers. At this point, the client can request more data from the server.

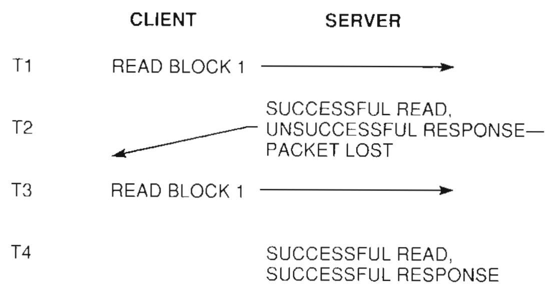

CLIENT
SERVER
T1
READ BLOCK 1
T2
SUCCESSFUL READ, UNSUCCESSFUL RESPONSE—PACKET LOST
T3
READ BLOCK 1
T4
SUCCESSFUL READ, SUCCESSFUL RESPONSE

Figure 7 Idempotent Request

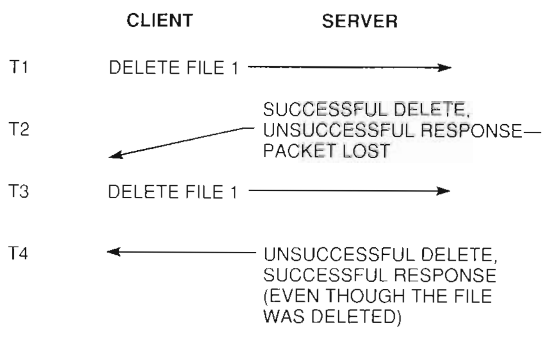

CLIENT
SERVER
T1
DELETE FILE 1
T2
SUCCESSFUL DELETE, UNSUCCESSFUL RESPONSE—PACKET LOST
T3
DELETE FILE 1
T4
UNSUCCESSFUL DELETE, SUCCESSFUL RESPONSE (EVEN THOUGH THE FILE WAS DELETED)

Figure 8 Nonidempotent Request

Response Buffering

The LAST protocol does not acknowledge the receipt of messages because it relies on the integrity of the underlying LAN to deliver datagrams without error. Consequently, the BMP layer must buffer all response data transmitted to the client to protect against packets that are lost or discarded. In such a case, the BMP layer transmits the original response message back to the client without sending the message to the server process.

For instance, consider the two cases shown in Figures 7 and 8. In Figure 7, a client generates a read request at time T1. The server processes the request and generates a response at time T2. The response is lost due to congestion, so the client requests the same data again, as indicated at time T3. The server rereads the file and generates a new response. Since the read operation is naturally idempotent, i.e., it can be repeated without changing the result, the operation completes successfully.

In the case depicted in Figure 8, we changed the operation from a disk read to a delete file. Here, the client makes the delete request at time T1, and the server successfully deletes the file at time T2. The response message is again lost. When the client reissues the delete file request at time T3, the server fails in its attempt to perform the operation because the file no longer exists. The delete operation is not idempotent; thus, repeating the operation yields a different outcome.

We cannot determine in advance the actual idempotency of any given request. Therefore, the BMP layer must cache all response buffers. If a response message is lost, the server transmits the original response message instead of retrying the entire operation. If, as in the second example, the server is able, at time T4, to transmit the actual buffer used at time T2 to store the response message, the operation can complete successfully.

To facilitate the buffering of response data, the transport provides a transaction identifier for request and response messages. This identifier is set by the client BMP layer whenever a new request is received from the redirector. The server stores this identifier and verifies it against the identifier of the next request. If a received request has a duplicate identifier, the request must be a retransmission and the server transmits the message in the cached response buffer. If the identifier is unique, the cached buffer is returned to the server by means of the shared queues, and a new request is created. The client's single-threaded nature ensures that the transaction identifier method is successful in detecting a retransmission.

NetBIOS Emulation

The PATHWORKS transport interface implementation relies on the request/response behavior of the DOS redirector. However, the redirector uses the standard DOS network basic I/O system (NetBIOS) interface to communicate with transports, and this interface does not exhibit request/response behavior. Therefore, our implementation is not a true NetBIOS emulation and can prevent common NetBIOS applications from operating correctly.

To resolve this problem, we developed a common NetBIOS interface between the DECnet and LAST transports. After receiving a request, the client first tries to connect over the LAST transport. If the connection attempt fails, the request passes to the DECnet transport. Thus, standard NetBIOS application requests operate over the DECnet transport; only redirector requests are processed over the LAST transport.

Final Benchmarks

At the completion of the project, we performed benchmark tests to measure server performance for varied workloads and for a directory tree copy. Table 2 shows the results for varied workloads. The first column of the table describes the test performed. ALL I/O represents a raw disk I/O test in which the measured client issues read and write requests of various buffer sizes ranging from 128 bytes to 16KB. TP represents a transaction processing test that measures random read and write requests of small units of data. This test emulates a typical database application. The workload value indicates the number of client systems used in the test to produce a background workload. As one might expect, as the workloads increase, the performance of the measured client degrades.

The entries in each row of the table are the elapsed time and percent CPU utilization for the given test. We measured server performance over the LAST protocol using our optimized interface and over the DECnet protocol using the standard VMS QIO interface. For the ALL I/O tests, the resultant elapsed time is the actual time it took to complete the test. For the TP tests, the performance numbers are the average of all the PCs tested. As Table 2 shows, we were able to decrease the elapsed time for each benchmark while maintaining the same or decreased CPU utilization.

The two graphs in Figures 9 and 10 illustrate these results. In the ALL I/O test, CPU utilization using the optimized interface increases steadily as the workload increases. Using the standard QIO interface, CPU utilization increases at a faster rate once a specified workload is reached. Although the TP graph in Figure 10 contains only two data points, it is evident that CPU utilization is proportionally higher for five workloads than it is for one. We performed multiple tests to verify that the results could be reproduced consistently.

The final benchmark test performed was a directory tree copy using the DOS XCOPY utility. In this test, the utility copies the directory tree first from the server to the client and then from the client to the server. The bottleneck in this test is known to be the file creation time on the server. Therefore, we expected a more efficient transport interface to

Table 2 Final Benchmark Test Results for Varied Workloads

		LAST Protocol		DECnet Protocol	
Test Description		Elapsed Time (seconds)	CPU Utilization (percent)	Elapsed Time (seconds)	CPU Utilization (percent)
All I/O	0 Workloads	840	4	961	4
All I/O	2 Workloads	943	69	1074	75
All I/O	4 Workloads	1091	100	1434	100
TP	1 Workload	59	39	79	50
TP	5 Workloads	163	83	212	93

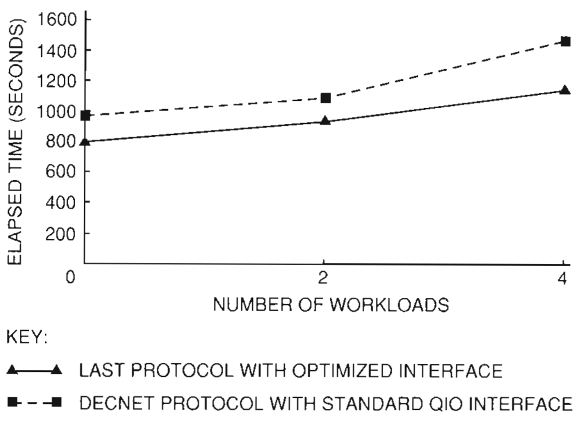


Figure 9 ALL I/O Test Results

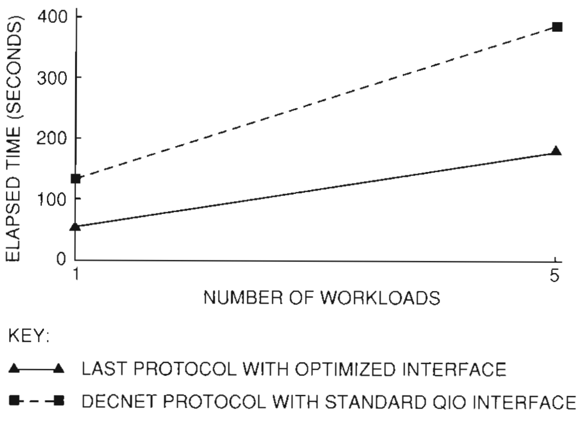


Figure 10 TP Test Results

Table 3 Final Benchmark Test Results for a Directory Tree Copy

Test Description	LAST Protocol Elapsed Time (seconds)	LAST Protocol I/O Rate (KB/sec)	DECnet Protocol Elapsed Time (seconds)	DECnet Protocol I/O Rate (KB/sec)
XCOPY to Client	115	39	15	39
XCOPY to Server	119	38	121	37

have no effect on server performance. The test results in Table 3 support our theory. The I/O rate and the elapsed time over both the DECnet protocol (using the standard transport interface) and the LAST protocol (using the optimized transport interface) are nearly the same.

Acknowledgments

I wish to thank Jon Campbell for incorporating the interface design modifications into the file server, Alpo Kallio for developing the client software, and Alan Abrahams for designing the combined DECnet/LAST NetBIOS interface and for his encouragement and support.

Anthony J. Rizzolo
Elizabeth A. Brewer
Martha A. Chandler

Design of the PATHWORKS for ULTRIX File Server

The PATHWORKS for ULTRIX product integrates personal computers with the ULTRIX operating system on a local area network. The software supports both the TCP/IP protocol and the DECnet transport stacks. The design and implementation of the PATHWORKS for ULTRIX file server is based on a client-server model. The server provides file, print, mail, and time services to client PCs on the network. Network file service management is accessed through a PC-style menu interface. The file server's performance was optimized to allow parallelism to occur when the client is generating data at the same time the server is writing the data to disk.

The PATHWORKS for ULTRIX file server connects industry-standard personal computers running Microsoft's server message block (SMB) protocol to Digital computers running the ULTRIX operating system. The server provides a network operating system for PC integration among users of the ULTRIX, DOS, and OS/2 operating systems.

The PATHWORKS for ULTRIX server provides file, print, mail, and time services to client PCs on the network. The software is layered on VAX systems and on reduced instruction set computer (RISC) hardware. It supports both the transmission control protocol/internet protocol (TCP/IP) and the DECnet transport stacks. The base product also provides centralized server-based management accessed through a PC-style menu interface.

In addition, the PATHWORKS for ULTRIX server implements a network basic I/O system (NetBIOS) naming service that allows clients on the network to obtain the DECnet node address of the server in the DECnet environment or the TCP/IP address of the server in the TCP/IP environment. The DECnet NetBIOS naming service conforms to Digital's specification for a DECnet NetBIOS interface. The TCP/IP NetBIOS implementation conforms to the requests for comment specifications, RFC 1001 and RFC 1002.[1,2]

This paper discusses the considerations for designing and implementing a PC local area network (LAN) server in an ULTRIX system environment. It describes the multiple process model and its component processes that coordinate management activities and server requests. It then presents our design of a management interface and our selection of a network interface. Finally, the paper describes the PATHWORKS file system, printing, performance considerations, and the server configuration.

Process Model

The process model selected for the PATHWORKS for ULTRIX server differed substantially from the process model chosen for the PATHWORKS for VMS product. The PATHWORKS for VMS server uses a single process model in which all client requests are processed by a single process, the VMS server. The PATHWORKS for ULTRIX server, in contrast, uses a multiple process model, in which one client is serviced by one server process.

Certain characteristics of the ULTRIX operating system environment determined the choice of a multiple server process model. First, the ULTRIX operating system constrains a process to 64 simultaneously open files. Therefore, with multiple server processes, each client connection is allowed access to 64 open files. In a single process model, a pool of 64 file descriptors is provided which limits access to 64 open files, regardless of how many clients connect. In addition, the multiple server process model has the advantage of being able to run in a multiprocessor environment.

Within the context of the multiple process model, we required a central administrative entity—the administration process—that would coordinate management activities and server requests. The administration process communicates with both the server and management processes through message queues. This process model is depicted in Figure 1 and is described in the following sections.

Administration Process

The administration process is known as pcsaadmd. As the central administrative entity, this process is responsible for initialization and start-up of the server, and for data management while the server is running. Starting the PATHWORKS for ULTRIX server is accomplished through execution of the administration process from within the rc.local file when the ULTRIX system is booted, or from the management menu when the management interface is run. Initialization of the server environment is necessary before any server management or connections can be established.

Initialization involves starting the NetBIOS process (pcsanbud), parsing the configuration file (lanman.ini), creating and initializing a shared memory segment, creating semaphores and a message queue, parsing the services database, clearing statistics, defining objects on the DECnet objects, and establishing signals. The main task of the administration process is processing requests from the management interface (pcsamgr) and file server processes (pcsafs). The initialization procedure occurs in the following sequence.

To simplify server start-up, the NetBIOS process is started from the administration process. At start-up, the NetBIOS process claims the server name and responds to name queries from clients during establishment of a session connection. It also provides for sending datagram and broadcast messages on the LAN. These two tasks are initiated by the user through the management interface by means of the Send and Broadcast Message functions. All management requests are processed through the administration process. Request handling is discussed in more detail later in this section.

The administration process parses the lanman.ini file to obtain server configuration parameters such as maximum number of sessions, connections, and open files. The administration process uses these parameters to establish the size of the shared memory segment it creates. The shared memory segment includes a session database, a connection database, a file database, common variables, and a locking database. Once shared memory is created, the administration process initializes it to a known state that includes clearing and date stamping the server statistics portion of the segment. The administration process creates semaphores to attain data integrity in the shared memory segment, since multiple file server processes read and write to memory.

The services database tracks file and print service creation from one execution of the server to another. This database is read at initialization, and the directories offered by the file service defined, as well as printer information, are verified.

The last step required at initialization is the creation of a message queue to process incoming requests from the management interface and file server processes. As said earlier, request processing is the main task of the administration process.

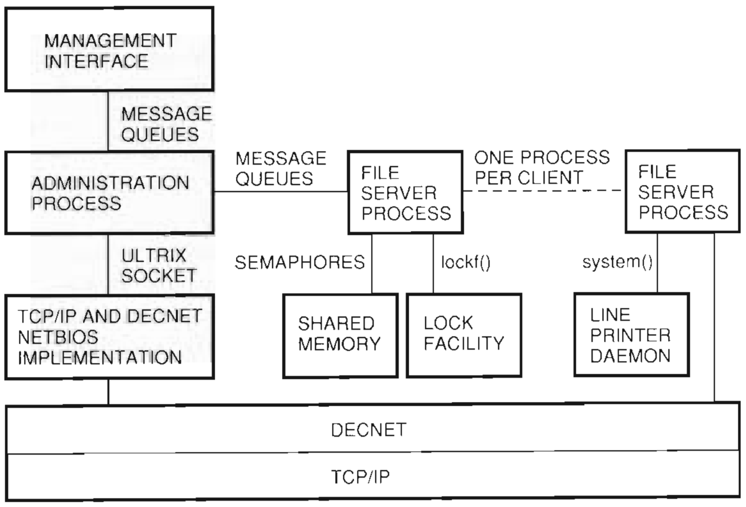


Figure 1 PATHWORKS for ULTRIX Process Model

Message queues are used as the interprocess communication mechanism. Early in the process development, we investigated other options: named pipes, sockets, and packet passing through shared memory. Only message queues offered administrative control. Initially, we used one response message queue for each file server process and one queue for the management interface. This was unacceptable because the default number of message queues on the ULTRIX system is 40 without reconfiguring the kernel. Therefore, we chose to combine the messages on one response queue from all the file server processes and retain a separate response queue for the management interface. Since the number of requests from file server processes is small, this method was acceptable. The administration process reads requests on one message queue and replies to a message queue defined in the message. The request queue is established with an ID known by all processes so they can attach to the queue at start-up. The administration process handles requests for session establishment and connection from file server processes as well as requests for system management/administration from the management interface.

File Server Process

The PATHWORKS for ULTRIX file server is started through one of two mechanisms, depending on which transport is used. The dnet_spawner process starts the file server process in a DECnet environment, and the inet_spawner starts the server in a TCP/IP environment. The server process is initially started as a root process, since it may need to run on behalf of several users. When a client issues a connection request, a server process is initiated. The server then sends a message to the administration process message queue requesting a session connection. After the session connection is granted by the administration process, the file server completes its initialization by connecting to shared memory and waiting for incoming client requests.

During the design phase of the multiple server process model, it became clear that using a slow interprocess communication mechanism has a detrimental impact on the overall performance of the server. For this reason, we decided to use shared memory for all time-critical shared data. Because the amount of shared memory is somewhat limited, all data that is not time critical is communicated across message queues. As can be seen in Figure 1, the file server and administration processes use shared memory as well as message queues for communication.

Since multiple processes can simultaneously update and access shared memory, a method was needed to guarantee data integrity. The methods chosen varied among the databases, depending on the type and speed of the access required to the database. Obviously, the easiest and also the slowest way was single-process management of access to shared memory. This worked well in the case of allocating connection data blocks, since the administration process had to be notified of connections. The open and read-write paths for the file and locking database, however, would be significantly affected by an incorrect decision. For this reason, we decided to protect these databases with an ULTRIX semaphore. In effect we single threaded all the paths through the open path as well as the locking update path. Use of this semaphore caused little or no degradation in performance. With our system processes and mechanisms established, we now had to consider the needs of the system administrator.

Management Interface

Our primary goal in designing a management interface for the PATHWORKS for ULTRIX server was to provide an application that could run unaltered on any type of terminal. The management interface also had to be consistent in presentation and manipulation of screens; and most importantly, it had to be easy to use when managing file and print services, workstation registration, and ULTRIX system users and groups. Other design considerations included performance, the ability to extend the functionality provided, and the ability to port the application to future platforms.

The management interface was designed to incorporate X/Open Curses software, which is a set of C library routines. X/Open Curses is provided by the ULTRIX operating system and is used to optimize screen management. X/Open Curses code uses the terminfo database, a collection of terminal definitions and characteristics that enables the application writer to perform terminal-dependent functions in a terminal-independent manner. Through X/Open Curses software and its use of the terminfo database, the PATHWORKS for ULTRIX management interface can support any type of terminal.[3]

The next step was to design an easy-to-use application that requires minimal knowledge of ULTRIX system management. We chose a PC-style format that uses pulldown menus, input forms, scroll regions for displaying information, and screen-sensitive help. Default input information is displayed whenever possible to provide sample data and to minimize the amount of input required.

The design of the management interface was structured into three layers: screen manipulation, data validation and presentation, and application programming interface (API).

Screen Manipulation

The first layer of the management interface is the X/Open Curses software. All screen manipulation routines reside at this level. X/Open Curses encompasses the implementation of reverse video attributes for highlighted text, cursor movement, window updates, and the creation of menus, forms, and scrolling regions. Any type of screen interaction is performed and managed by this layer of code. As a result, the screen manipulation layer is portable to any environment in which X/Open Curses is supported.

Data Validation and Presentation

At the data validation and presentation layer, data obtained from the screen interface is validated. The data is then packaged and processed by the API layer. Information returned by the API layer is unpacked and formatted for screen presentation.

Application Programming Interface

The API layer is responsible for all communication with the administration process. The management interface does not store or manipulate server management data directly. Instead it makes requests of the administration process in the form of APIs through message queues. Each request requires a response and does not complete until a response is received.

Network Interface

When designing an application that must communicate on a network, one of the important decisions is how to control access to the network. The Berkeley Software Development version 4.3 of the UNIX kernel, upon which the ULTRIX operating system is based, provides two network interfaces.

The first network interface is known as the socket interface. It uses a socket structure to identify the endpoint of an ULTRIX network connection. Under the ULTRIX system, the socket interface is the primary interface to the network.

The second network interface in the ULTRIX system is the X/Open transport interface (XTI). This transport service interface is not restricted to either the DECnet or the TCP/IP transport. A common interface to the network allows either transport to be accessed transparently. With XTI the communication endpoint is identified by a local file descriptor. On the ULTRIX system, the XTI interface is provided through a library that converts the XTI calls into socket calls. Since performance was one of our primary concerns, we decided to use the socket interface because it connects directly to the ULTRIX operating system.

Multiple Transport Support

In order to support both the TCP/IP and the DECnet transports, we needed to overcome the differences between a message-based protocol (DECnet) and a stream-based protocol (TCP/IP). With a message-based protocol, data received from the network arrives in compact packets. With a stream-based protocol, message boundaries are not preserved; the data flows in a stream. Since Microsoft's SMB protocol is a message-based protocol, the server needs to re-create these message boundaries. As a result, the server must identify the transport provider. This information is provided by the socket layer when the server process is started. The server can re-create the message boundaries by combining this information with message size information provided in the TCP/IP NetBIOS header. With the message boundary information, the server can re-create the message. The C pseudocode fragment in Figure 2 shows the instructions to re-create message boundaries.

PATHWORKS File System

The PATHWORKS file system provides an application layer that attempts to emulate the DOS file system. Several trade-offs and restrictions were required in order to implement this file system on the ULTRIX file system. This section describes these trade-offs and restrictions and explains our design choices.

File Name Mapping

The file name space in the ULTRIX system is not restricted to the 8.3 naming format supported by DOS. DOS limits file names to eight characters followed by an optional period and an optional three-character extension. This is referred to as DOS 8.3 file name format. DOS file names are uppercase characters and are case insensitive. Under the ULTRIX

```
/* SMBptr - Pointer to SMB netbios header */
/* rdlen - Number bytes read from network */
/* BytesRcvd - Bytes already received     */
/* BytesLeft - Bytes left in current message */


rdlen=read(network,SMBptr);
BytesRcvd=rdlen;
BytesLeft=sizeof(netbios header);
BytesLeft+=ntohs(EXT16(SMBptr->nb.length)-bytes_rcvd;

/* We will wait until we receive all the data in the msg */
/* before we terminate this loop. This loop will only be */
/* entered if we are running TCP/IP. */

while (BytesLeft!= 0) {
  rdlen=read(network,&SMBptr[BytesRcvd]);

/* If we don't get any data it means the client must have */
/* torn down the session so abort */
/* our session. Note AbortSession() must exit and*/
/* not return here.*/

  if (rdlen<=0) AbortSession();

/* Update the counters to account for what we just read */

  BytesRcvd+=rdlen;
  BytesLeft-=rdlen;
}

/* If this is a SESSION_REQUEST message, then send the ACK*/

if (SMBptr->nb.type == SESSION_REQUEST) SendSessionAck();

/* If this is a SESSION_MESSAGE, then handle the SMB */

if (SMBptr->nb.type == SESSION_MESSAGE) DispatchSMB();
```

Figure 2 Receiving Stream Data Code Fragment

system, the file name is a 32-character string in which the period (.) is a legal character. The ULTRIX file system is case sensitive and supports both uppercase and lowercase characters in the file name.

To resolve this incompatibility between operating systems, we mapped the DOS file name space into the ULTRIX file name space. DOS, being case insensitive, views the world of file names in uppercase, but ULTRIX file names are typically lowercase characters. We chose to map all DOS file names to the equivalent lowercase name. Any file on the host ULTRIX operating system that meets our criteria, i.e., lowercase names and 8.3 format is visible to the DOS client.

This approach was suitable in all environments except International Standards Organization (ISO) 9660 CD-ROM file systems. These file names conform to the DOS uppercase, 8.3 file naming format. When the file server determines that one of the services is on an ISO 9660 CD-ROM file system, the file-name mapping algorithm is changed to allow only uppercase file names that follow the DOS 8.3 format.

DOS Attribute Mapping

The DOS file system provides file attributes that do not necessarily map to ULTRIX file attributes. The challenge was to preserve these DOS attributes within the ULTRIX file system without impacting the host system user who might also be sharing the file. The DOS attributes consist of read-only, hidden, archive, and system.

The DOS read-only attribute maps directly to the ULTRIX directory attributes mask. If the write attribute is turned off under the ULTRIX system, the files change to read-only status.

The DOS hidden attribute specifies that a file should not be displayed on a normal directory search/lookup. We mapped this bit to the ULTRIX set user ID bit.

The DOS archive attribute specifies that a file has been changed since the last time the archive attribute was set. It is generally used by the backup program to determine which files have changed since the last backup. We mapped the archive attribute to the ULTRIX set group ID bit.

The DOS system attribute specifies a special system file that is normally not displayed on a directory listing, and in some cases is not backed up. We mapped the DOS system attribute to the Owner eXecute bit. If this bit is set, the server cannot include these files on a normal directory search, unless requested.

Byte Range Locking

The most noticeable difference in byte range locking between the ULTRIX operating system and the DOS operating system is that byte ranges under the ULTRIX system are purely advisory. Advisory locking works as a mechanism to signal that a byte range is currently in use. The ULTRIX system, however, does not enforce the locks; therefore it is possible to read/write a byte range that is locked simply by ignoring the lock.

On the other hand, DOS has mandatory locking. If a byte range is locked, the user can neither read nor write a locked byte range. We needed to convert the ULTRIX lock manager into a mandatory lock manager from the DOS clients' point of view. To do this, the ULTRIX lock manager has to check for a lock on a byte range on every read or write from the file server. If any portion of the byte range is locked, the client receives a lock failure message.

In the initial release of the server, we believed that the standard ULTRIX lock manager would provide enough performance and granularity to allow DOS client software to function correctly and quickly. We learned that this was not always the case. For example, in a network file system (NFS) environment, additional time for granting or denying the lock request was needed to resolve a lock on the network. In addition, the ULTRIX lock manager viewed the byte range as a signed integer, but the DOS lock manager viewed the byte range to be locked as an unsigned integer. This disparity led to problems with applications that used byte range locks with the sign bit set to provide synchronization for database updates. We found that the ULTRIX lock manager was deficient in the DOS client environments. For these reasons, we decided to write a private lock manager for applications that could not use the ULTRIX lock manager.

To resolve locking problems among these applications, we designed a private lock manager for the PATHWORKS for ULTRIX server. We provided a high-performance lock manager that could lock byte ranges used by DOS applications. In other words, the server lock manager would treat the lock range as an unsigned number instead of a signed number. We also provided the option of passing the lock information to the ULTRIX lock manager for those applications that needed this functionality.

Open File Mode Locking

The DOS client provides a mechanism for controlling access to opened files. It allows the client who initially opens a file to control access to the file by other clients. The DOS client allows files to be opened in one of four modes:

- DENY_NONE mode allows all types of files to be opened by all users.
- DENY_READ mode allows other users to open the file for writing but not reading.
- DENY_WRITE mode allows other users to open the file for reading but not writing.
- DENY_READ_WRITE mode does not allow other users to open the file.

The ULTRIX operating system, on the other hand, has only two modes for a shareable file lock. The first is SHARED_ACCESS mode, which allows multiple readers and writers and is therefore equivalent to the DENY_NONE mode. The other mode is EXCLUSIVE_ACCESS mode, which does not allow multiple accesses to the same file and therefore is equivalent to DENY_READ_WRITE mode under DOS.

Because of these differences, we attempted to map the two modes not covered by the ULTRIX file lock manager, the DENY_READ and DENY_WRITE modes. After some investigation, we decided mapping was not necessary. If a file was opened in one of these two modes, we specified that the ULTRIX server should open the file in ULTRIX SHARED_ACCESS mode. We reasoned that an ULTRIX application that was cooperating with a DOS application would not use these two modes to open the file since they are not available under the ULTRIX system. Obviously these two modes need to be supported among DOS-based PCs on the server. Each time a user opens a file, the list of currently opened files is scanned to enforce the open mode and to be sure that the ULTRIX operating system conforms to the DOS interpretations of these modes. Therefore, only the half deny modes being passed through to the operating system are not enforced. This design decision should pose no danger to applications sharing data.

Directory Search Implementation

The DOS file search algorithm and the SMB messages that provide support for directory searches were difficult to implement on the ULTRIX file server. The core SMB protocol provides only two states for a search context, begin new search and continue a previous search. However, the server needs to be informed that the client has completed a directory search context. Then the server would be able to free local data associated with the search context. The implementation of this SMB posed two challenges: how to control the amount of memory required and how to map a search continuation identifier.

To minimize the amount of memory required to maintain search contexts, we designed a table of search context structures that contains a local timing value. If the table becomes full and a block (structure and time value) needs to be reused, the oldest block is deemed reusable. This approach efficiently manages the unpredictable memory requirements of an SMB search.

The search continuation provides a directory information structure which contains a four-byte field that determines the point at which the search is to continue. This four-byte field is well suited to the ULTRIX file system. The gnode field, a longword, can be used for the four-byte field's search continuation ID. Given this ID, the server has the ability to parse the contents of the directory until it finds a file with a matching gnode; it then continues the search from that point.

PATHWORKS for ULTRIX Printing

In addition to file services for DOS and OS/2 system-based clients, PATHWORKS for ULTRIX provides print services for these PC clients. Our design objective was to allow the PC clients access to all the functionality on the native ULTRIX print queue in a transparent manner. A second objective was to implement the functionality provided by NET PRINT, the client utility for printing, on the native ULTRIX line printer daemon (LPD).

Although the LPD provided all the basic printing capabilities, it did not provide timed scheduling of print jobs. To enable timed scheduling, we added the /AFTER switch to the server through a mechanism within the ULTRIX operating system. When a /AFTER switch is detected in one of the extended printing SMBs, a batch job is run at the time specified in the print request.

The ULTRIX print spooler provides spooling for all types of printers, e.g., those attached locally as well as network printers and reverse Local Area Transport (LAT) printers connected to PCs. Reverse LAT printing is very important in our environment because most PCs have printers attached and most installations have a need to share those printers among several PCs.

The ULTRIX print spooler provides print filters which translate files to various printers. Print filters conceptually sit between the LPD and the actual file to be printed. During printing, the LPD reads a "printcap" file to determine if a print filter is associated with this queue. The print filter is started in a forked process with its standard output device (stdout) pointing to the printer and its standard input device (stdin) pointing to the input file stream. The print filter is responsible for converting the file from the input stream (stdin) into a device-specific output that is usable by the printer (stdout). This feature allows the PATHWORKS for ULTRIX server to support printing on a wide variety of third-party printers.

The design of the ULTRIX printing subsystem enabled the PATHWORKS for ULTRIX server to provide an interface to many different printers and printer configurations easily and efficiently.

Performance Considerations

As part of the design process, we observed the performance of the file server during interactions with the client. We needed to compare various conflicting alternatives and their effects on the overall performance of the server. Some of the alternatives we studied were the advantages of using the ULTRIX system cache versus implementing our own cache. We also studied the issue of persistent lock requests on the network and the server. These alternatives are discussed in this section.

File I/O

Since the ULTRIX operating system provides a kernel-based, disk cache mechanism, we designed the operating system's cache manager to perform all caching globally. The cache manager updates the cache buffers, performs read ahead on data streams, and flushes the cache buffers from data written to disk.

The file server performs disk writes as write behinds. When a request to write data is received from a client, the server responds by acknowledging success before the write is attempted (assuming the client has proper write access to the file). This optimization allows parallelism to occur between the client and the server because the client is generating more data at the same time the server is writing the data to disk. If the write fails, however, the server notes that the last write failed and returns the error on any subsequent access to the file.

Heuristics

We found that certain applications would continually flood the server with lock requests even though the lock requests kept failing. These persistent lock requests from applications used valuable CPU time on the server system as well as network bandwidth. For this reason, the ULTRIX server needs to determine if a client is being persistent and continually requesting locks which are failing. When the server detects continuous lock requests, it delays the lock request for a random period of time and then checks if the lock has become available. The server then either grants access if the lock is available, or returns the error at that time. This procedure reduces lock request traffic, since most locks are of short duration.

Security

Connection requests between client and server require a security check. Since PATHWORKS for ULTRIX was designed to be layered on the ULTRIX operating system, we were able to take advantage of its security features. When a client attempts to connect to the server, a username and password can be passed as part of the connect message. If these are supplied, the user is validated through system calls to obtain the password file entry for that user. If the user is not found in the /etc/passwd file or if the password is invalid, the user is denied connection. If the ULTRIX system is running in enhanced security mode, further checks are made to ensure the account has not been disabled or the password expired. In either of these cases, the connection would be denied. If a username is not supplied, a default guest account may be used to establish privileges.

VAX versus RISC Considerations

During the development of the PATHWORKS for ULTRIX file server, we did not anticipate that our code would have to differentiate between VAX and RISC architectures. We expected that code written for an ULTRIX system in a RISC environment would be recompiled on a VAX system. For the most part, our assumptions were correct, except in the areas of memory allocation.

On the VAX system, shared memory maps directly *after* the data segment in memory. This implementation prohibits the allocation of memory *above* a shared memory segment. In the RISC implementation, shared memory is allocated at the very top of the memory image; therefore a great deal more memory can be allocated before the bottom of the shared memory segment is reached. The difference in shared memory allocation between the RISC and VAX systems is shown in Figure 3.

To increase the data segment size in the VAX system, we replaced all malloc()calls in the server modules with the following pseudocode:

```
Disconnect from shared memory malloc()
Reconnect to shared memory
```

Since this code is required only in a VAX environment, it is compiled when the server is built.

PATHWORKS Server Configuration

The PATHWORKS for ULTRIX file server allows the system manager to configure the server environment to make the most efficient use of shared memory. The following parameters included in the lanman.ini file are the determining factors that enable shared memory to be scaled.

- maxsessions: The maximum number of PC workstations that can be simultaneously connected to the PATHWORKS for ULTRIX server.
- maxconnections: The maximum number of connections PC workstations can make to the services offered.

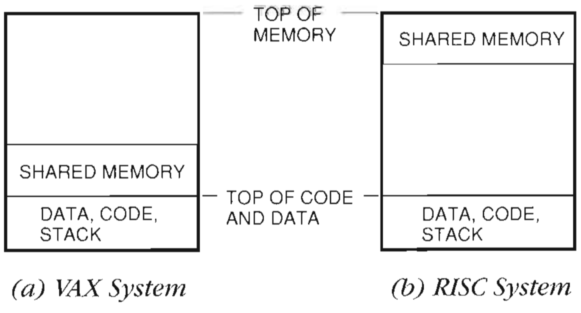


Figure 3 Image Memory Layout

- maxopens: The maximum number of files the server can have open simultaneously.
- uniqueopenfiles: The maximum number of unique open files the server can have open simultaneously.
- maxserverlocks: The maximum number of byte range locks the server can lock simultaneously.

To help the user apply these parameters to a particular system, the "pcsa memory" command acts as a shared memory sizing calculator. It allows the user to input the parameters and then either indicates that the new parameters will fit in the current system, or that the system shared memory parameters need to be changed to support the new configuration.

Information Logging

PATHWORKS for ULTRIX information logging was designed for the ULTRIX system manager as well as writer/users of the LAN Manager application. It provides event and error logging information in two distinct formats. The first format uses the ULTRIX system log file: syslog. This log file is typically monitored by ULTRIX system managers. All processes which comprise PATHWORKS for ULTRIX submit configuration information and error conditions to this file. The file server process also logs information regarding service usage, sessions, and connections.

The second form of event logging is performed entirely by the server process. The server process logs error and audit events to LAN Manager-compatible log files: error log and audit log. These log files are accessible through the management interface as well as through the LAN Manager API interface provided with DOS and OS/2 LAN Manager implementations. These files contain information on session logon/logoff, password errors, connections started/rejected, resource access granted/denied, and server status changes.

Summary

The PATHWORKS for ULTRIX file server, together with the PATHWORKS for DOS and PATHWORKS for OS/2 products, provides a distributed computing environment. The file server is based on a client-server model that offers transparent access to ULTRIX system resources from PC clients. It provides the necessary tools to integrate these two diverse computing environments in a manner that is both efficient and easy to manage.

Acknowledgments

Many people were involved in the design and building of the PATHWORKS for ULTRIX file server from its inception to its shipment. We wish to thank all those people: Paul Messier and Jim Flaherty, who guided our efforts; Dan Smith, who designed and implemented the NetBIOS layer; Ken Cardinale, who wrote the product documentation; Marlene Steger, who ensured that the product shipped on time; and the many individuals who successfully brought this product to market.

References

1. *Protocol Standard for NetBIOS Service on a TCP/UDP Transport: Concepts and Methods,* Internet Engineering Task Force (IETF) RFC 1001 (March 1987).

2. *Protocol Standard for NetBIOS Service on a TCP/UDP Transport: Detailed Specification,* Internet Engineering Task Force (IETF) RFC 1002 (March 1987).

3. *ULTRIX-32 Guide to Curses Screen-Handling,* ULTRIX Document Set, Software Development, vol. 2 (Maynard: Digital Equipment Corporation, Order No. AA-MF07A-TE, 1988).

Mitchell P. Lichtenberg
Jeffrey R. Curless

DECnet Transport Architecture

The PATHWORKS family of software products includes an implementation of the DECnet transport protocol to allow Intel-based personal computers access to network resources. This implementation, the DECnet Network Process (DNP) transport component, provides basic file and print services, terminal emulation, and application services. The new DNP component for the version 4.1 release of the PATHWORKS for DOS client software is written in assembly language to improve performance and reduce memory usage. The DOS and OS/2 versions of the component contain the same base source code, thus decreasing the development and maintenance costs.

Digital's PATHWORKS family of software products provides the means to integrate personal computers into the Digital network environment. The PATHWORKS for DOS client software includes device drivers, network transports, utility programs, and applications that allow PCs full access to the resources available in local and wide area networks (LANs and WANs). Transparent file sharing, electronic mail, and terminal emulation are examples of services supported by PATHWORKS client software.

The DECnet protocol suite is implemented in Digital's standard set of software for interconnecting VAX and reduced instruction set computer (RISC) systems. DECnet software, which is included in the PATHWORKS client software, enables PC integration. The DECnet protocols allow PATHWORKS products to use the infrastructure of existing Digital networks and to provide common utility programs and network management capabilities.

However, integrating PCs into a network system presents many design challenges to software developers. They must provide network access without limiting the functionality of the PCs and without compromising the compatibility of the existing PC software and peripherals. Since the PC architecture has limited memory resources and few built-in features for networking, PC network software architectures must be as transparent as possible, reducing memory usage and emulating local peripherals and software interfaces.

To implement this transparent architecture, the PATHWORKS products comply with PC-related industry standards. Most such standards result from popular vendor software applications or hardware. For example, Microsoft's LAN Manager software product influenced the acceptance of the industry-standard server message block (SMB) protocol. This session layer protocol, implemented over a variety of transports, is used in the LAN Manager redirector for transparent file sharing and peripheral emulation. Digital licenses the LAN Manager software in order to provide these services as features of the PATHWORKS product family. Digital extended the LAN Manager across a LAN or a WAN system by using the DECnet transport protocol as the transport layer in its PATHWORKS products.

In this paper we first present our rationale behind the design of the DECnet transport component in PATHWORKS for DOS version 4.1, as well as in PATHWORKS for OS/2 version 2.0. We then describe the new component's internal structure, follow a typical network operation through the component, and compare this version of the software component with previous versions.

PATHWORKS Client Software and the DNP Component

Since its initial release, the PATHWORKS product family has implemented the DECnet transport protocol to provide access to basic file services and printer sharing, terminal emulation, and application services. This network software implementation is called the DECnet Network Process (DNP) transport component. Figure 1 illustrates the relationship between the DNP transport component and the other memory-resident PATHWORKS client software components.

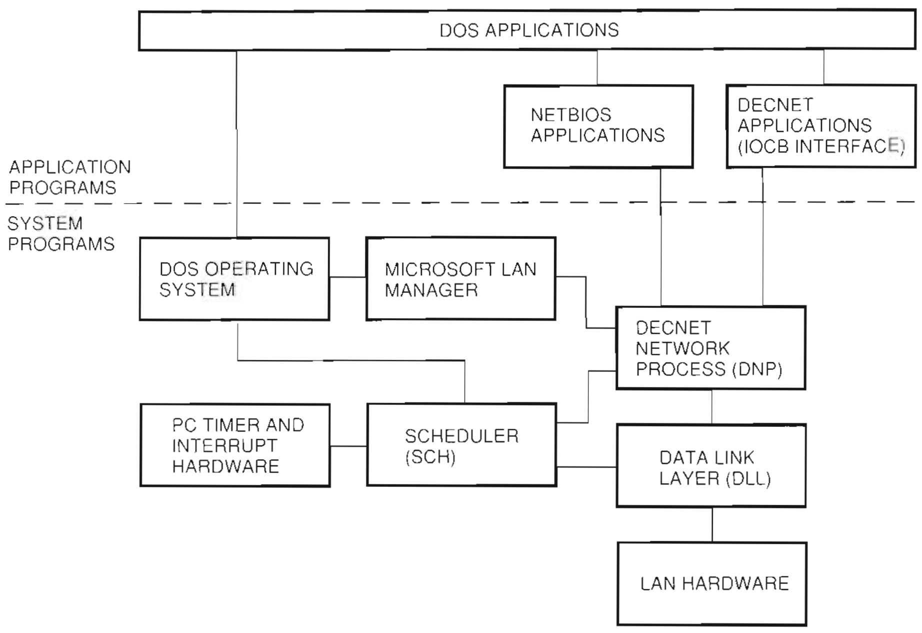


Figure 1 PATHWORKS Client Components

Goals for PATHWORKS Client Software

PC network software products are judged primarily on two criteria: performance, usually measured with popular benchmark programs, and resident memory usage, a limited resource that may restrict other applications. Increasing performance and decreasing memory usage are major goals for all new releases of the PATHWORKS client software. In the PATHWORKS version 4.1 client software, Digital sought to double the performance of the DNP transport component for small data transfers, while decreasing the size of the code by 50 percent. Another goal was to significantly reduce maintenance costs in order to free engineering resources for future project development.

Before describing how we went about achieving these performance, memory, and development cost goals in PATHWORKS version 4.1, we review the functionality of the DECnet DNP implementation. We also discuss the component in relation to other PATHWORKS client components to give the context in which our design decisions were made.

The DNP Component Functionality

Application programs can use DNP transport services through one of two software interfaces: the network basic I/O system (NetBIOS) interface and the I/O control block (IOCB) interface. The widely accepted NetBIOS interface is used by applications and drivers that comply with industry-standard specifications to provide peer-to-peer transport services on a LAN. The IOCB interface is specific to Digital's DECnet transport implementation of the DECnet protocols. IOCB provides a socket interface similar to the one used by the ULTRIX operating system. This IOCB interface is used by DECnet-specific application programs.

To communicate with the network, the DNP transport component calls the data link layer (DLL) software interface. The DLL component is used by all PATHWORKS client components to send and receive packets on the LAN. This component demultiplexes incoming packets based on their protocol type (e.g., local area transport [LAT], local area system transport [LAST], or DECnet transport) and delivers these packets to the corresponding PATHWORKS client component. The DLL component also transmits packets on the LAN, either directly or indirectly by calling standards-based network drivers. To reduce memory consumption, the DLL component provides a global buffer pool that the DNP and other transport components can use for building network packets or for storing unacknowledged data.

To provide timing and background process services, the DNP component calls the PATHWORKS real-time Scheduler (SCH) component. The SCH communicates directly with the DOS operating system and the PC's timer and interrupt hardware to create a simple cooperative process environment. This environment includes sleep and wake calls, and periodic interval timers. The functions of the SCH component are similar to those performed by true multitasking operating systems, such as the OS/2 system, which use preemptive scheduling.

Considerations for a New DNP Component Design

In previous PATHWORKS client software, separate teams implemented and maintained the DOS and OS/2 versions of the DNP transport component. We decided to use the same base source code for both versions and thus reduce development and maintenance costs. We then proceeded to consider our design options.

Originally, the DNP component was written in the C programming language. The internal architecture remained basically unchanged during the five years following its release. This stable code should have been easy to port between operating systems. However, the internal architecture of the OS/2 system was never considered in the original design because this system was not available until 1988. Retrofitting the DOS version of the DNP component for the OS/2 operating system was difficult, and we were not able to maintain a common source code base.

To achieve the performance, memory, and development cost goals described earlier in this section, we considered the following three approaches:

1. Rewrite the current DNP transport component
2. Write a new DNP transport component in C
3. Write a new DNP transport component in assembly language

Rewriting the current DNP component would not have produced a desirable amount of code common to the DOS and OS/2 versions. In addition, this approach would have resulted in only minimally improving the maintainability of the code. Writing a new transport component in C would have yielded a more portable code, but the performance and memory usage would not have compared favorably with other vendors' transports. We decided to write the new transport component in assembly language to make optimal use of the limited memory available on today's personal computers.

PATHWORKS Version 4.1 DNP Transport Component Design

Internally, the DNP transport component comprises various modules that correspond approximately to the layers of the DECnet protocol suite, as shown in Figure 2. Later in this section, we describe the major DNP modules and how they cooperate.

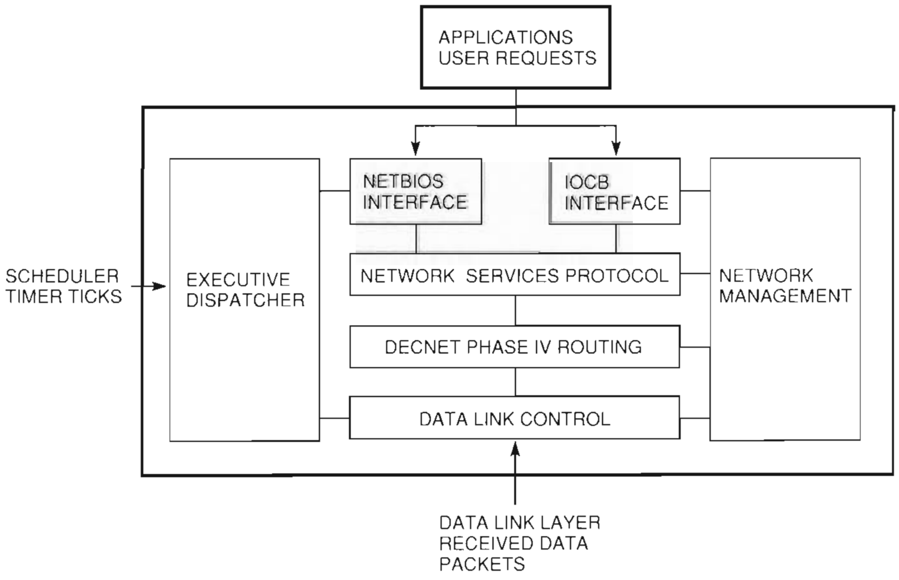


Figure 2 Internal Architecture of the DECnet Network Process Component for PATHWORKS Version 4.1

Three types of events can cause the DNP component to respond or to "wake up": user requests, received packets, and timer ticks. All of these events are asynchronous, since they are generated by hardware interrupts or user actions that are not managed by the operating system. Each time the DNP component processes an event, variables and data structures internal to the component change. In designing the component, we had to ensure that the events would not interrupt one another.

To protect the data structures in a generic way, all versions of the PATHWORKS DNP component use a queuing system called the executive. Asynchronous events are queued to the executive module, where a semaphore guards the dispatching and processing routines. The queue and the semaphore guarantee the following: the receipt of a new event does not interrupt ongoing processing, and events are processed in the order in which they arrive.

Under the DOS operating system, the main loop of the executive module uses the PATHWORKS SCH component to "sleep," process pending events, and sleep again. Events that arrive while the main loop is executing are simply placed on the queue. Operating under the DOS system, on which no background processing services exist, the DNP component uses the PATHWORKS SCH component. Since the OS/2 operating system does provide a background processing environment, the corresponding version of the DNP component uses the native background processing and scheduling functions of the OS/2 operating system to perform the same tasks.

Data Structures

The DNP transport component uses three primary data structures to manage network links and to transfer data: the request (REQ) data structure, the link status block (LSB) data structure, and the large data buffer (LDB) data structure.

To queue events for processing, the REQ data structure is allocated from a global pool. Pointers to a user request or to network data are stored in the REQ structure and then placed on the executive dispatcher queue. The REQ structure is also used to describe unacknowledged data and to store events in the event log. Using the same pool for different purposes saved memory and decreased the overall complexity of the component. Figure 3 illustrates a typical request queue to the executive dispatcher.

The executive module reads each event, i.e., collection of messages or user requests, from the request queue and dispatches the event to the appropriate handler routine, according to event type. The routine then further dispatches the event to specific subroutines to handle the individual messages or requests. The lowest-level routines keep network links active and transfer data to and from the remote system.

In previous versions of the DNP component, the REQ data structure consumed 48 bytes of memory. We reduced its size to 22 bytes by creating variant records that contained only those data fields necessary to identify the type of request.

The LSB data structure maintains the current status of a logical link. In addition to the network services protocol (NSP) variables, the LSB structure stores other data, including the queue of unacknowledged data and the queue of outstanding transmit and receive requests. Figure 4 illustrates the contents of the LSB and associated data structures for an active logical link.

The user requests are attached to queues on the logical link. For storage of unsent or unacknowledged data, the DNP component uses a REQ data structure to point to an LDB data structure. The LDB

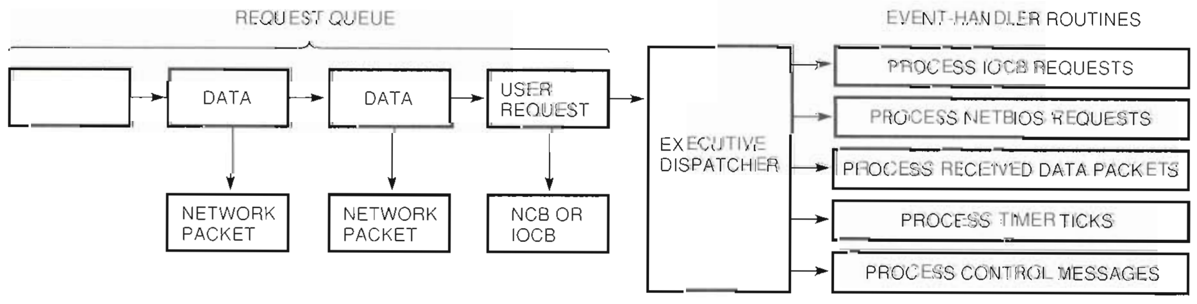


Figure 3 DNP Executive Dispatcher Module and Incoming Request Queue

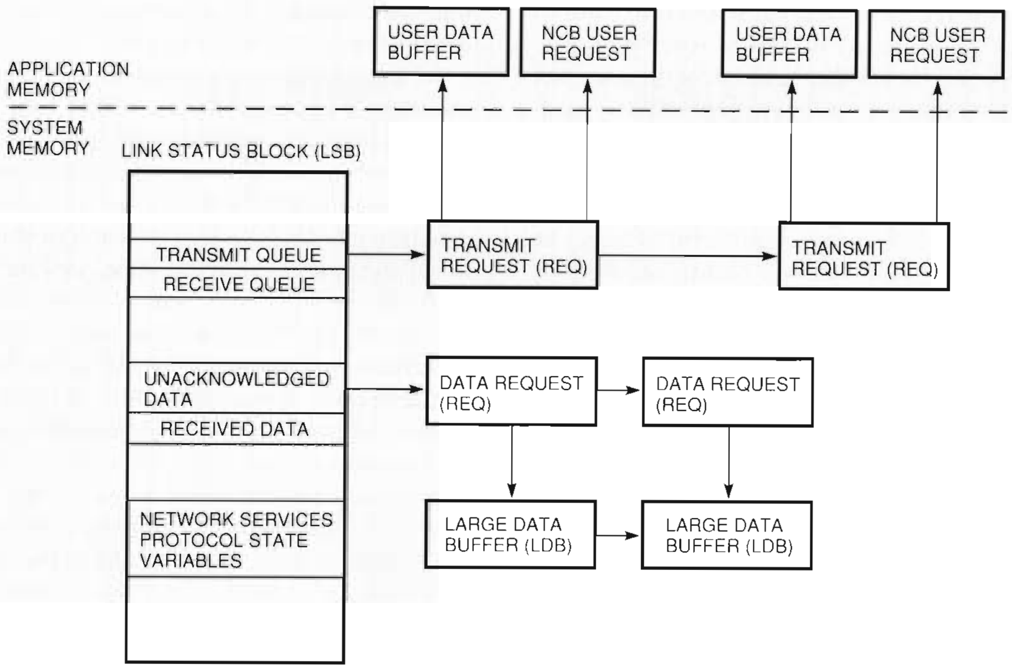


Figure 4 Link Status Block and Associated Data Structures

structures belong to the Ethernet or token ring data link component and are shared by other protocols. Before transmitting data, the DNP component allocates first an LDB data structure and then a REQ data structure that points to the LDB. The REQ structure is placed on the outgoing message queue of the LSB structure, and the NSP layer eventually transmits the REQ data.

Internal DNP Modules

The DNP transport component consists of various modules. We now describe the data link control (DLC) module, the NSP module, and the NetBIOS and IOCB modules.

The DLC module is responsible for communication with the Ethernet or token ring data link component. Only the DLC module calls the data link, and the differences between the DOS and OS/2 versions are hidden in the DLC module to present a consistent software interface to the rest of the DNP component.

To make the NSP and DECnet Phase IV routing modules as operating-system independent as possible, we developed a simplified software interface to communicate with the Ethernet or token ring DLC module. The DLC module calls the data link that is specific to the operating system. Providing the software interface allowed us to use common code for all of the modules that do not directly access the data link.

The NSP module manages the transport protocol, including the buffering, flow control, and error recovery mechanisms. In PATHWORKS version 4.1, we changed the buffering and flow control algorithms to match more closely the types of traffic that PC network applications are likely to generate.

Most users of the NetBIOS interface post receive requests before transmitting a request for data from a server. Some implementations of the NetBIOS interface do not buffer received or transmitted data inside the transport component, so applications must prepare to receive before requesting data from the server. To best manage the incoming data, the DNP component of PATHWORKS version 4.1 uses XON/XOFF flow control for NetBIOS logical links and segment flow control for logical links that use the IOCB interface. The previous version used segment flow control for both the NetBIOS and IOCB interfaces. XON/XOFF flow control causes fewer messages to be transmitted on the wire, especially in request/response session layer protocols, and is most successful when the receiving node has a buffer ready to accommodate the incoming data. Segment flow control is more robust and allows the DNP component to better regulate the rate of incoming data. This method of flow control can be especially useful for non-request/response protocols such as those used in the DECwindows software.

The NetBIOS and IOCB modules form the session layers for the DNP component. In previous versions of the DNP component, the NetBIOS module was layered on top of the IOCB interface. However, as we mentioned earlier in the paper, most popular network applications use the NetBIOS interface. We decided to increase the performance of those applications by designing the new DNP component in such a way that the NetBIOS module directly calls the NSP module.

We recognized another element of the DNP design that could be improved. Earlier DNP versions copied the user's NetBIOS request into a local data structure for easy access. The extra resources required to store and copy the user requests diminished the overall performance of the DNP component. To improve performance, the DNP component now stores a pointer to the original user's request and manipulates the request directly.

NetBIOS compatibility is one problem that many vendors face when writing network transport components. The NetBIOS software interface is defined in several different specifications, and many applications depend on quirks and bugs in the design. The PATHWORKS NetBIOS interface must emulate these bugs completely for certain applications to work properly. We paid careful attention to the bug reports from customers in previous versions of the PATHWORKS software when rewriting the NetBIOS layer to provide complete compatibility.

A Typical Network Operation

To illustrate the sequence of events through the DNP component for a typical network operation, consider the transmission of 64 kilobytes (KB) of data through the NetBIOS interface. The application that wishes to send the data constructs a NetBIOS control block (NCB) data structure and submits it to the NetBIOS software interface. The DNP component receives control, creates a queue entry for the NCB structure, and then wakes the SCH component. Waking the SCH component causes the main loop of the DNP component to begin execution. The executive module checks the request type and dispatches the entry to the NetBIOS module where the transmit request is placed on the logical link's transmit request queue. The transmit request initially points to the user's NCB and the beginning of the user's data buffer.

The NSP module copies data into the LDB data structures and queues these LDBs onto the unacknowledged data queue. The amount of data copied depends on the size of the transmit pipeline, which is a network management parameter. Each time data is copied into an LDB data structure, the pointer advances in the transmit request queue. When all of the data is copied into the LDBs, the user's transmit request is completed, allowing the application to continue execution while the DNP component transmits the queued data.

If the flow control mechanism permits sending data, the NSP module calls the routing layer to add routing headers. The data link control module then transmits the packets on the LAN. The remote network system responds with acknowledgment messages, which are placed on the request queue and processed when the DNP component returns to the main loop. An acknowledgment message causes the LDBs to be returned to the data link control module and makes space available on the transmit pipeline. The NSP module can then refill the transmit pipeline by copying more user data into the LDB data structures and restart the transmit process.

Results

We achieved our project goals for the DNP transport component in PATHWORKS version 4.1 client software. The new design allowed us to reduce memory usage, improve performance, and reduce maintenance cost.

Memory Usage

We reduced the resident size of the DNP component from 53KB to 33KB. The reduction in the size of the internal data structures freed up enough memory resources to allow the DNP component to handle over 200 concurrent network links. Previously, the DNP component was limited to 64 links.

Performance

By coding in assembly language, and optimizing the path for sending data messages to the network, performance was nearly doubled for small data transfers. Small data transfers account for the majority of transfers in database applications.

The graph shown in Figure 5 represents DECnet performance, measured in messages transferred per second, as a function of message size, ranging from 64 to 65,500 bytes. We include data for versions 4.0 and 4.1 of the DNP component. As the message size increases, the curves converge because the Ethernet adapter's performance becomes the limiting factor for throughput. Smaller message sizes are typical in many industry-standard PC benchmark programs and applications.

The benchmark program used to calculate DECnet performance transfers data from one PC to another as fast as possible, using fixed message sizes. The hardware used in these tests consisted of 20-megahertz Intel 80386 microprocessors with 16-bit DEC EtherWORKS Turbo (DE200) adapters running on a private Ethernet segment.

Maintenance Costs

Debugging the common source code base for the DOS and OS/2 versions is much simpler than for the previous version of the DNP component. Since the OS/2 version uses the memory protection features of the PC's Intel microprocessor, invalid memory references under the OS/2 version cause system traps that would not have been detected under the DOS version. Nearly 90 percent of the code is common to the DOS and OS/2 versions of the DNP component. The number of source lines was reduced from 73,000 (the DOS version only) in PATHWORKS version 4.0 to 53,000 (the DOS and OS/2 versions combined) in PATHWORKS version 4.1. Rewriting the component also improved its compatibility with third-party NetBIOS applications.

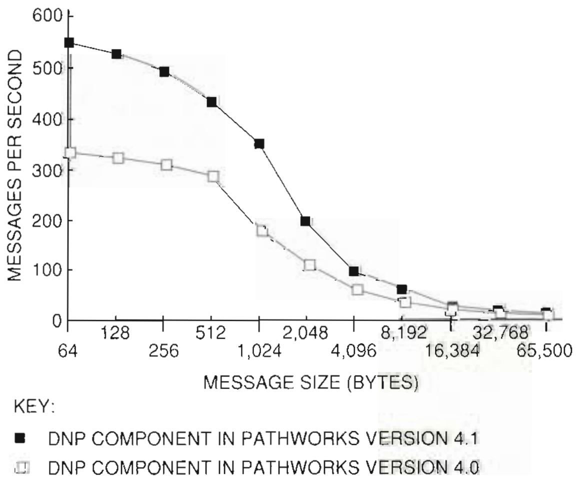


Figure 5 DECnet Network Process Component Throughput

Debugging features were added to the DNP component in PATHWORKS version 4.1 that allow customers to adjust their DECnet configuration easily and precisely. The DNP component now collects statistics on the maximum number of REQ, LSB, and LDB structures allocated, and on the size of each pool. Using this feature, we found that the version 4.0 DNP component allocated nearly twice as many REQ data structures as it needed under normal client workloads. As a result, we lowered the default allocations to further reduce memory consumption.

Conclusion

The DECnet transport component project for the version 4.1 release of the PATHWORKS client software was a success; we came very close to our original goals for memory, performance, and software development costs. In addition, many of the techniques, algorithms, and data structures used for this effort can be applied to future network transport development.

General References

IBM NetBIOS Application Development Guide (Armonk, NY: International Business Machines Corporation, 1987).

Microsoft/3Com Network Driver Interface Specification, version 2.0.1 (Redmond, WA: Microsoft Corporation, 1990).

PATHWORKS Programmer's Reference, version 4.1 (Maynard, MA: Digital Equipment Corporation, 1991).

DECnet Phase IV General Description (Maynard, MA: Digital Equipment Corporation, Order No. AA-N149A-TC, 1983).

Microsoft MS-DOS Programmer's Reference (Redmond, WA: Microsoft Corporation, 1990).

Microsoft OS/2 Device Driver Reference (Redmond, WA: Microsoft Corporation, 1989).

Andrew W. Nourse

Microsoft Windows Network Virtual Device Drivers in PATHWORKS for DOS

Digital's PATHWORKS for DOS version 4.1 personal computer integration software includes two network virtual device drivers for the Microsoft Windows environment. These drivers allow Windows applications operating in a protected processor mode and standard DOS applications in a virtual machine to concurrently access services designed to run in real mode under the DOS operating system. The network virtual device drivers, available only in Microsoft Windows enhanced mode, manage DECnet and NetBIOS operations and permit the full use of these interfaces.

Microsoft Windows virtual device drivers are loadable software modules that extend the Windows operating system and enable it to support peripheral devices, memory resources, and software applications. Some of these modules allow applications that operate in different processor modes with corresponding differences in memory access to communicate with one another in a network system. Digital's PATHWORKS products make it possible to integrate personal computers into local or wide area network systems. The PATHWORKS for DOS software includes two network virtual device drivers, which manage DECnet and network basic I/O system (NetBIOS) operations in the Microsoft Windows environment for PCs.

This paper begins with a discussion of the Microsoft Windows environment for which the PATHWORKS for DOS product provides network virtual device drivers. The basic processor operating modes and the Microsoft Windows operating modes are described, preparatory to an explanation of Microsoft Windows enhanced mode. This explanation is essential because virtual device drivers operate only in enhanced mode.

Next, the paper details the capabilities of virtual device drivers, such as providing the means for Windows and DOS applications to communicate. The focus then turns to the environment for developing Microsoft Windows virtual device drivers and concludes with a description of the structure and functionality of the two network device drivers included in the PATHWORKS for DOS software.

Microsoft Windows Environment

The Microsoft Windows environment is a graphical, multiapplication system for personal computers that use the Intel 80286 or higher microprocessor. For 80286-based systems, the Windows system operates in its standard mode, using the real and protected processor modes. On the 80386 or higher microprocessor, the Windows system can also operate in its enhanced mode, using both protected and virtual processor modes. Enhanced mode allows the Windows system to fully utilize processor features such as virtual memory and multiple virtual machines. Virtual device drivers are available only in this enhanced mode.

Basic Processor Operating Modes

All members of the 80x86 family, including the 80386 microprocessor, calculate addresses in memory by using a segment register and an offset. However, the method for calculating the physical address varies, depending on the processor mode. The basic processor operating modes are real mode, protected mode, and virtual mode.

Real Mode This mode is used by the DOS operating system exclusively and by most DOS applications. The processor calculates physical addresses by shifting the contents of a 16-bit segment register left by 4 bits and adding a 16-bit offset. Therefore, only the first 1 megabyte (MB) plus 65,519 bytes of a PC's physical memory are directly accessible in this mode.

The basic layout of PC memory is shown in Figure 1. The first megabyte of physical memory is known as conventional memory. This area may include the PATHWORKS implementation of the DECnet transport protocol, namely the DECnet Network Process component, as well as other memory-resident software. In addition, conventional memory may contain the DOS operating system and DOS applications. The next 65,519 bytes are called the high memory area. Bank-switched memory, known as expanded memory, may also be available. In real mode, memory protection and virtual memory are not available, illegal instructions are generally ignored, and I/O instructions are always allowed.

Protected Mode In this mode, a segment register contains a selector. Part of the selector is an index into a descriptor table maintained by the hardware. A flag in the selector indicates which of two descriptor tables to use, the local descriptor table or the global descriptor table. The processor adds the offset to the linear address obtained from the appropriate descriptor table. The 80386 implementation differs from that of the 80286 because the 80386 processor offers both 16- and 32-bit general registers and offsets, whereas the 80286 processor has 16-bit general registers and offsets.

In protected mode, if paging is disabled, the linear address is the physical address. If paging is enabled, the linear address is decoded into a page directory entry, a page table entry, and an offset. The page directory entry identifies a page table, and the page table entry provides a physical address.

Protected mode is used by applications that use DOS extenders to access memory beyond that which is accessible from real mode. 80386 processors operating in protected mode may use virtual memory. In this mode, an illegal instruction causes a processor trap, and I/O instructions may be selectively allowed or trapped.

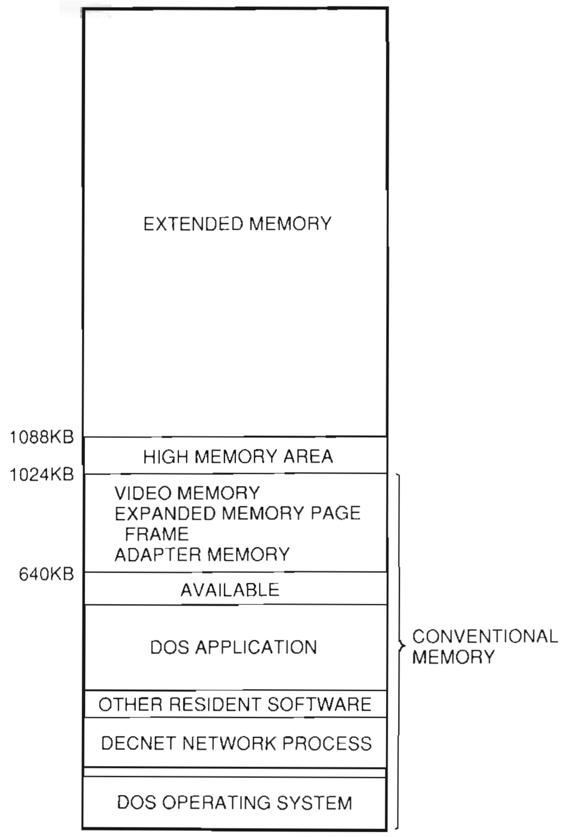


Figure 1 Basic PC Memory Layout

Virtual Mode This mode implements a virtual machine that emulates the behavior of an 8086 microprocessor. Address calculation in this mode is similar to that in real mode, except that in virtual mode the result of the shift-and-add operation is a linear address. The processor converts this address to a physical address, as in protected mode. Processors operating in virtual mode may use virtual memory. Also, each virtual machine can have a separate page directory, an illegal instruction causes a processor trap, and I/O instructions may be allowed or trapped.

Microsoft Windows Operating Modes

The Microsoft Windows environment supports several operating modes.

Windows Real Mode Similar to previous versions of the Windows system, Windows 3.0 can operate in real mode, i.e., use conventional memory, expanded memory, and the high memory area. This mode is not supported in Windows 3.1.

Windows Standard Mode Windows 3.0 and 3.1 can operate in standard mode on the 80286 or higher microprocessor. This mode uses the protected processor mode, but does not take advantage of the 32-bit features of the 80386 processor. The Windows system and Windows applications are located outside conventional memory, except for code necessary to provide the communication links with DOS and other resident software.

Standard DOS applications run in real mode and occupy the full screen, as if the Windows system were not present. Switching between Windows and non-Windows applications is accomplished by performing a sequence of keystrokes in exactly the same manner as under the MS-DOS version 5.0 task switcher. Virtual device drivers are not used in standard mode.

Windows Enhanced Mode In enhanced mode, the Microsoft Windows system provides each non-Windows application a virtual machine in which to operate. These machines are preemptively multitasked, so even compute-bound, non-Windows applications can run in the background. The Windows system and all Windows applications share a single virtual machine so they can communicate with each other.

The Microsoft Windows system uses the protected and virtual modes of the 80386 processor. Paging is always enabled. The first 1MB plus 65,519 bytes of the linear address space is mapped to the first 1MB plus 65,519 bytes of memory belonging to the virtual machine currently executing. This mapping allows each DOS application its own block of memory in which to run.

Some data must be shared among the virtual machines. Whenever all or most of the data in a page is shared, a global page is used. Most resident software that was loaded before the Windows system start-up is stored in global pages. Selected data within these global pages may be maintained separately for each virtual machine. This practice is called instancing and may be requested by the resident software.

To support operations requested by virtual machines, virtual device drivers extend the Microsoft Windows kernel. The drivers are loaded at Windows initialization and effectively become part of the kernel.

The Microsoft Windows enhanced mode kernel uses 32-bit registers and offsets. The segment registers are loaded with selectors that allow access to all of memory when the kernel is operating and eliminate the need to break code and data into 64-kilobyte (KB) segments of memory. This memory model is known as the flat model.

Although the Windows enhanced mode kernel is written to use 32-bit registers and offsets, most of the remaining libraries supplied with the Windows system and nearly all applications are written to use 16-bit registers and offsets. The Windows applications run in protected mode, whereas virtual mode provides support for the DOS applications, which need not even be aware that the Windows environment exists.

Virtual Device Driver Capabilities

Virtual device drivers provide the means for Windows and DOS applications to communicate, support asynchronous operations, virtualize hardware ports and interrupts, and directly handle hardware and software interrupts. These capabilities are described in the following section.

Communication between Protected-mode and Real-mode Software Applications

A virtual device driver provides a bridge between Windows applications running in protected mode and DOS terminate and stay resident (TSR) applications written to run in real mode with no knowledge of protected mode. A Windows application that calls an application programming interface (API) passes it a valid protected-mode address. Without virtual device drivers, the real-mode software would interpret this address as a real-mode address, usually pointing to a location within the DOS operating system. A virtual device driver can map the memory into conventional memory and change the addresses so that the real-mode software correctly accesses the caller's data. The virtual device driver should enter a critical section to avoid task switching while calling real-mode software that is not reentrant.

Communication between Transient DOS Application Software and Global Resident DOS Software

Most DOS application software and DOS TSR software is not designed to run in the Microsoft Windows environment. While executing, a DOS application is mapped into conventional memory. If the application calls resident software, and a task switch occurs while an operation is in progress, data would be delivered to the wrong application.

There are two ways to handle this situation. The virtual device driver can enter a critical section to disable task switching until the operation is complete. This approach works well for synchronous operations that never take a perceptibly long time to complete.

However, the system does not respond to most user input while the virtual device driver is in a critical section. Consequently, for long synchronous operations, the end user of the application may believe that the system is hung. If the real-mode software supports asynchronous operations, the virtual device driver can convert the operation to an asynchronous call. Handling the situation in this manner requires that a critical section be entered only for the time it takes to queue the call, and then only if the real-mode software is not reentrant.

Support for Asynchronous Operations

Asynchronous operations, whether in real or protected mode, require that the virtual device driver be able to buffer data in a memory pool that is mapped into every virtual machine. In addition, the driver must set up a completion callback routine to wake up the virtual machine that made the request, deliver the data to that virtual machine, and transfer control to a caller-specified callback routine, if necessary.

Virtualization of Hardware Ports and Interrupts

Another function of virtual device drivers is to virtualize hardware ports and interrupts so that the Windows system can successfully emulate several 8086-based machines at once. Each virtual machine runs a DOS application that assumes it has sole use of a machine. DOS is a minimal operating system and does not provide much of the functionality required by applications. Therefore, most DOS applications bypass the operating system except to access the file system. It is common for an application to set up its own interrupt handlers and to read and write hardware ports. If several applications in separate virtual machines were to attempt these operations at the same time, the applications would interfere with one other. A virtual device driver can trap access to hardware I/O ports and regulate access to the actual hardware.

Direct Handling of Hardware or Software Interrupts

The virtual device driver can provide the functionality of real-mode software. If the user has no need to run this software outside the Windows environment, the software can be removed from memory. Removing the real-mode software reduces the need for context and mode switching, mapping, and copying, and thus offers a considerable performance advantage. If the resident software is removed, more memory is then available to DOS applications running in the Windows environment.

Development Environment

The Microsoft Windows system includes virtual device drivers. Microsoft also has a device driver development kit specifically for developing virtual device drivers.[1] This section describes the environment for developing and debugging this driver software.

Development Tools

Currently, virtual device drivers are written in assembly language because higher-level language compilers generally lack the ability to generate code with 32-bit offsets and registers. A special 32-bit assembler and linker are provided with the Microsoft Windows device driver development kit.

Debugging Tools

Virtual device drivers are debugged using the WDEB386 software module. This debug tool requires that a terminal or equivalent be connected to one of the communication ports on the PC; the debugger performs its I/O to that communications port. Symbols are available in the debugger, but source-level debugging is not provided.

To take full advantage of the WDEB386 capabilities, the debug version of the Microsoft Windows WIN386.EXE module should be used. This version contains many features essential for investigating the behavior of the Windows system and, in particular, for debugging virtual device drivers. The features include commands to display the registers, the stack, and the control blocks for each virtual machine. Many of the virtual device drivers included with the Windows system, and the two included in the PATHWORKS for DOS product, have a debug entry point that may be invoked by entering the period keyboard character, followed by the name of the virtual device driver. Two particularly useful debug entry points are .VMM and .V86MMGR, which provide detailed information about memory usage for each virtual machine, including the use of expanded memory and the high memory area. WDEB386 can be used successfully in the Windows environment to debug virtual device drivers and to diagnose bugs in the read-only memory basic I/O system (ROM BIOS) and other resident real-mode software.

The CodeView for Windows debug tool is intended for debugging applications and dynamic link libraries, not for debugging virtual device drivers. However, the CodeView and WDEB386 tools can be used simultaneously to diagnose problems that occur when applications cause the Windows system to fail.

The Network Virtual Device Drivers

The PATHWORKS for DOS software provides two APIs for task-to-task network communications. One is a DECnet socket-based interface, which uses an argument block called an I/O control block (IOCB). The other is the industry-standard PC networking interface, NetBIOS, with some extensions provided by Digital to support wide area networks. The NetBIOS interface uses an argument block called the NetBIOS control block (NCB). Both interfaces are fully supported in Windows enhanced mode.

Digital's PATHWORKS for DOS version 4.1 includes two virtual device drivers to support networking: VDNET.386, which handles DECnet socket calls, and VNETBIOS.386, which handles NetBIOS calls. Although they support different APIs, these two virtual device drivers are similar in structure. The discussion in this section applies to both drivers unless otherwise noted. These drivers are included with the current PATHWORKS version 4.1 product and with Windows version 3.1. To identify Digital Equipment Corporation as the developer of the drivers, Microsoft requested that the module names VDNET.386 and VNETBIOS.386 be changed to DECNET.386 and DECNB.386, respectively, in Windows version 3.1. In this paper, the nomenclature VDNET and VNETBIOS is used to refer to these two modules.

The drivers invoke the real-mode network software in the virtual machine that requested the operation. Creating a "network virtual machine" to which the driver would route all network activity would have allowed most of the network software to be loaded into a single virtual machine and thus freed up conventional memory for non-Windows applications. However, using this design would have incurred the overhead of switching on virtual machines for every network access, timer tick, and network hardware interrupt. In addition, creating a network virtual machine would have required that the data link layer and the DECnet scheduler be capable of performing the virtual machine switch. Finally, this design would be practical only for those users who access the network exclusively while operating in a Microsoft Windows environment.

Initialization

Virtual device drivers are called several times during Windows initialization. While the Windows system is still operating in real mode, the VDNET and VNETBIOS modules check to see if the resident network software is loaded. If it is not, there is no reason to load these drivers. A value is returned that aborts the loading of the drivers but directs the Windows system to continue loading.

After the Windows system enters protected mode, the drivers are called again during each successive phase of initialization. Each virtual device driver takes control of the software interrupts used for its respective API, reserves space in the control block of each virtual machine, obtains parameters from the SYSTEM.INI file, and allocates a pool of global memory for communication with the real-mode resident networking software. Figure 2 illustrates a system virtual machine and a virtual machine running a DOS application. The figure shows the pool of conventional memory that the virtual device driver allocates as global memory.

The drivers perform a "sanity check" to verify that the virtual device driver can distinguish global memory from memory that is local to a single virtual machine. However, the Windows function to perform this check can fail when running on some common unsupported software configurations. At this point, if the sanity check fails, the driver displays a message to advise the user to exit the Windows system.

Virtualization of the Network APIs

When an application issues a software interrupt for a DECnet or NetBIOS call, the appropriate virtual device driver gains control. If the application making the call is in protected mode, the virtual device driver always maps the call in memory. Otherwise, the driver software checks the control block (i.e., the IOCB or the NCB) and the buffer addresses to determine if they are stored in global memory, i.e., mapped identically in every virtual machine. If so, the virtual device driver does not map the call, because it will execute properly without mapping.

API Mapping If the control block and the buffer addresses are not stored in global memory, mapping is necessary. The virtual device driver

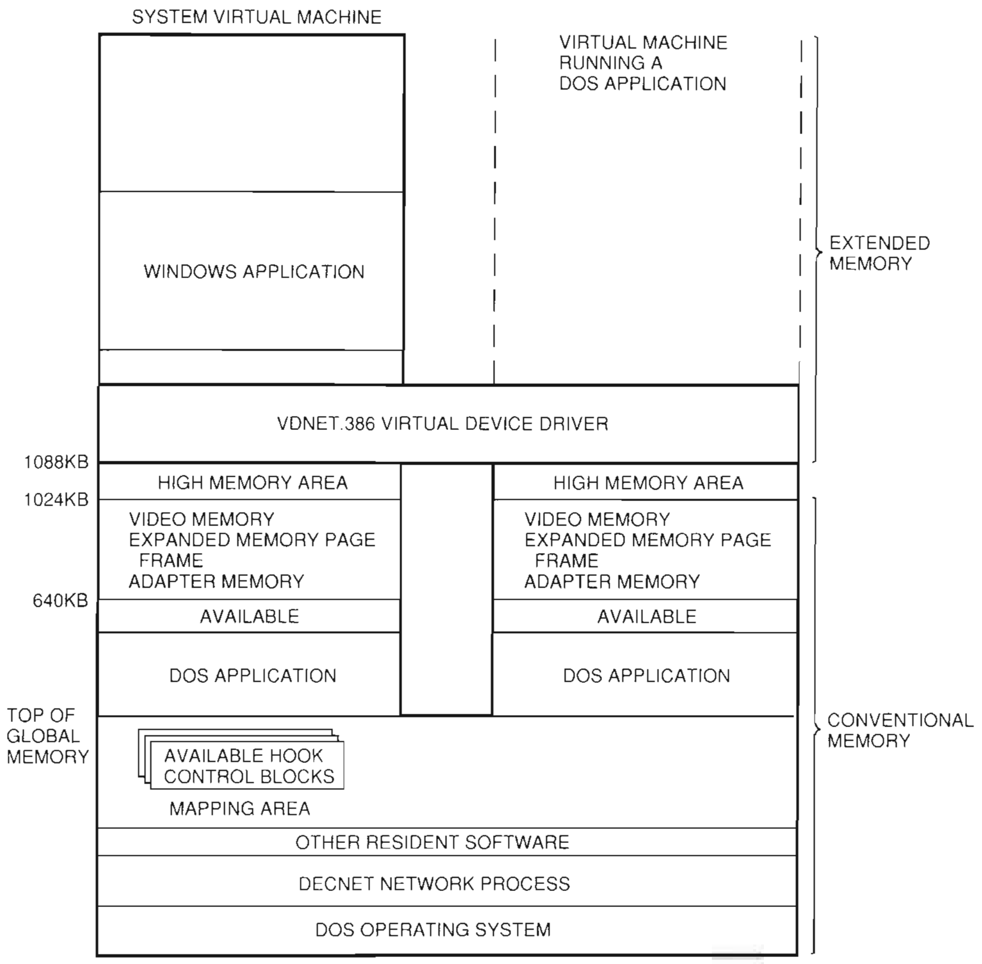


Figure 2 Microsoft Windows Initialization

allocates a hook control block to the operation. This control block resides in global memory and includes an IOCB or NCB, which the virtual device driver passes to the resident networking software. The driver globally maps the caller's buffers in the mapping-space pool allocated at initialization. The IOCB or NCB embedded in the hook control block contains addresses changed to point to the remapped address in the mapping-space pool. The callback (post) address is set to the callback routine in the virtual device driver, so the driver is called when the operation is complete.

Optionally, if the operation is a blocking call that takes a long time to complete, the virtual device driver may convert the operation to an asynchronous call. In this case, the driver sets an internal flag, HF_Suspend_Until_POST, and does not return control to the calling application until the operation is complete. All other virtual machines continue to run while the network I/O is in progress. This design prevents the operation from monopolizing the entire system.

Asynchronous Calls If the call is asynchronous or has been converted to an asynchronous call, the virtual device driver must establish a callback in order to be notified when the call completes. Because the virtual device driver runs in protected mode and the resident network runs in virtual mode, a special type of callback is required. The virtual device driver uses the Windows Allocate_V86_Callback service to obtain a real-mode pointer to an instruction in global memory that causes a trap when executed in virtual mode. The Windows system handles this trap and transfers control to the virtual device driver in protected mode.

Invoking the Network Process The virtual device driver is now prepared to pass the call to the real-mode networking software. The driver enters a critical section to avoid reentrance problems and calls the Simulate_Real-Mode_Interrupt service to invoke the network process as if it were being invoked in real mode. The virtual device driver leaves the critical section when the simulated interrupt returns. If the operation is not asynchronous, the caller's IOCB or NCB is updated, buffers are unmapped, and the hook control block is freed. Figure 3 shows a Microsoft Windows call to the network, intercepted by the virtual device driver and passed to the network process.

Callback Routine The device driver checks the HF_Suspend_Until_POST flag to determine if the call was a blocking call that the virtual device driver converted to an asynchronous call. If so, control must not return to the calling application until the operation is complete. Normally, the callback routine in the driver is called at this time. However, certain NetBIOS error conditions cause the operation to return immediately without calling the callback routine. Therefore, the NetBIOS virtual device driver checks the status of the call.

If the call is still in progress, the requesting virtual machine relinquishes its allocated time and retries when the process wakes up. This design protects the process from being awakened prematurely by another virtual device driver. Also, some NetBIOS request errors cause the NetBIOS software interrupt to return immediately and do not transfer control to the callback routine. Ordinarily, the process is only awakened by the callback routine in the virtual device driver on completion of the call.

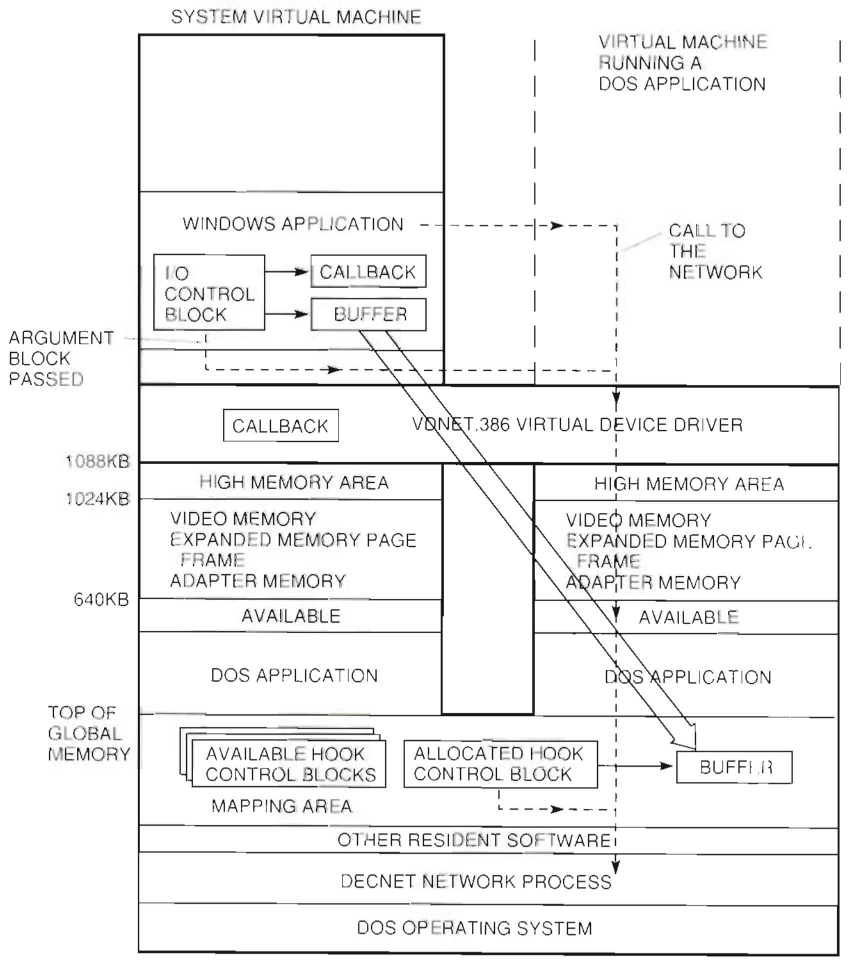


Figure 3 Invoking the Network Process

The Suspend_VM service can be used to block a virtual machine during such a call. However, suspending a virtual machine requires that the system call every Windows virtual device driver to notify it of the suspension. The notification process constitutes a high-overhead operation and is therefore unsuitable for this use.

If the operation is asynchronous, the system transfers control to the virtual device driver callback routine when the operation is complete, as shown in Figure 4. This routine calls the Windows scheduler to wake up the virtual machine that requested the operation. The Windows event services are also called to invoke the event-handler routine in the virtual device driver when the requesting virtual machine is scheduled. In this way, the virtual device driver regains control. This process restores the caller's context before updating the caller's data.

As shown in Figure 4, the event routine updates the user's argument block and calls the user's callback routine. Finally, the virtual device driver unmaps the buffers, frees up the hook control block, and returns control to the calling application.

Virtual Machine Termination

When a virtual machine terminates, all virtual device drivers are called to perform cleanup. The network virtual device drivers check for outstanding network operations to the virtual machine that is being terminated. All such operations are canceled, and a warning message is displayed to the user. Windows applications execute in the system virtual machine, so their outstanding network

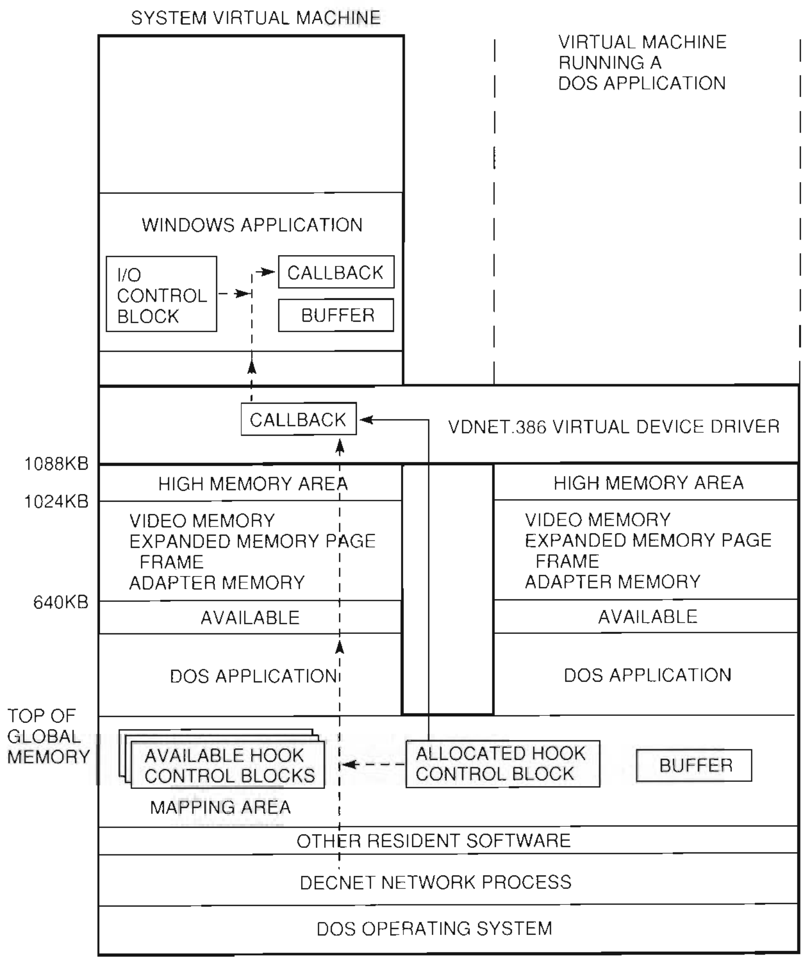


Figure 4   Callback Routine

operations, if any, are canceled when the user exits from the Windows system. Network operations by resident software are not canceled when a virtual machine terminates.

Debugging Entry Points

The VDNET and VNETBIOS network virtual device drivers provide debugging entry points for use by the Windows kernel debugger. These entry points give a formatted display of the hook control block for each hooked network call in progress. The display includes the requested function, buffer address, the handle of the virtual machine from which the call was requested, the virtual-machine-specific address of the caller's argument block, and flags. The flags included in the debugging display indicate the state of the operation, as shown in Table 1.

Special API Entry Point

The VDNET network virtual device driver provides an API entry point that allows application software to determine what version of the VDNET driver is loaded. This function is available to both protected-mode and real-mode applications.

Summary

PATHWORKS network virtual device drivers extend the Microsoft Windows enhanced mode environment to support most hardware that can be installed in a personal computer. These drivers also support all software that can run under the DOS operating system, including software that bypasses the operating system to access the hardware directly. Network virtual device drivers make network services available to the Windows kernel, to Windows and non-Windows applications, and to other virtual device drivers. The virtual device drivers included in the PATHWORKS for DOS software product permit full use of the DECnet and NetBIOS APIs, including Digital-specific extensions to the NetBIOS interface, in the Microsoft Windows enhanced mode environment.

Reference

1. *Microsoft Windows Device Development Kit—Virtual Device Adaptation Guide* (Redmond, WA: Microsoft Corporation, 1990).

General References

Intel 80386 Hardware Reference Manual (Santa Clara, CA: Intel Corporation, 1987).

Intel 80386 Programmer's Reference Manual (Santa Clara, CA: Intel Corporation, 1987).

Intel 80386 System Writer's Guide (Santa Clara, CA: Intel Corporation, 1987).

Table 1 Flags Included in the Debugging Display

Flag	Indication
HF_Wait_For_IRET	Cleared when the DECnet Network Process component returns to the virtual device driver.
HF_Wait_For_POST	Set if the virtual device driver callback is required; cleared when the virtual device driver callback is called.
HF_Wait_For_Sim_POST	Set if the caller requested callback; cleared when the caller's callback returns.
HF_POST_Crit	Set while in a critical section.
HF_From_PM	Set if the caller was in protected mode.
HF_Canceled	Set if the operation was canceled.
HF_Canceling	Set if the operation is being canceled.
HF_Suspend_Until_POST	Set if the operation is a blocking call that is being simulated using an asynchronous call. Do not return to caller until the operation is complete.

Dennis G. Giokas
Andrew T. Leskowitz

eXcursion for Windows: Integrating Two Windowing Systems

Digital's eXcursion for Windows display server is an application for Microsoft Windows. The eXcursion for Windows product is based on the X Window System and allows X client applications to display output, receive input, and exchange data in the Microsoft Windows environment. The eXcursion software visually integrates the X and Microsoft Windows environments—applications from both environments display on a single screen and have the same appearance. A key component of Network Applications Support (NAS) and Digital's PC integration program, the eXcursion for Windows display server enables information exchange among PC users and non-PC users linked throughout a network.

The eXcursion for Windows software is a display server based on the X Window System version 11, release 4 protocol and implemented as an application for Microsoft Windows software. eXcursion allows X11 client applications based on any X11 toolkit to display output and receive input in the Microsoft Windows environment. The two window environments are seamlessly integrated. Microsoft Windows software provides the window management for X Window System applications. The eXcursion display server smoothly handles the display and user input for the X applications along with data exchange between the applications.

This paper first establishes the relationship of the eXcursion display server to the X Window System and Microsoft Windows environments. It then presents the personal computing integration philosophy behind the development of the eXcursion product. This paper then relates the design philosophy and implementation architecture of the server. It concludes with a discussion of resource usage.

Overview

The DECwindows architecture integrates the user and graphical interfaces of the VMS, ULTRIX, and DOS operating system and desktop environments. The X Window System client-server architecture, on which the DECwindows system is based, provides the means to achieve this integration. The X architecture, as implemented by Digital's DECwindows Motif program, is shown in Figures 1 and 2. This architecture is hardware and software system independent. It allows X applications, or clients, to execute on any processor and display on any device in a distributed network.

X applications are linked with toolkits and libraries that include windowing controls, user interface objects, and graphics capabilities. The X toolkits also include interprocess communications capabilities that provide data interchange between the application clients. Figure 2 presents some of the libraries in the DECwindows environment.

These applications communicate with an X Window System display server over a network through the X protocol. The X protocol is independent of

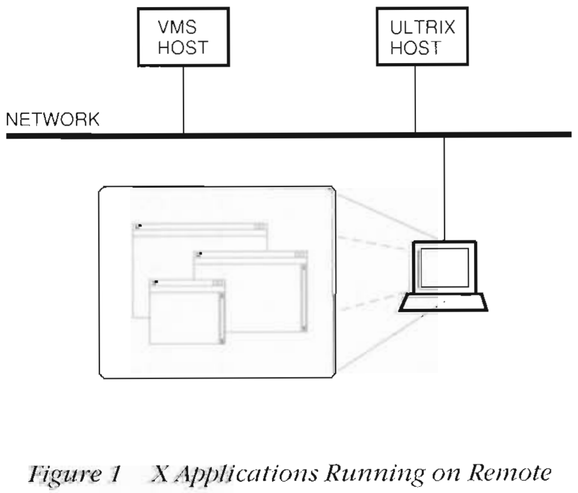


Figure 1 X Applications Running on Remote Nodes and Displayed on a PC

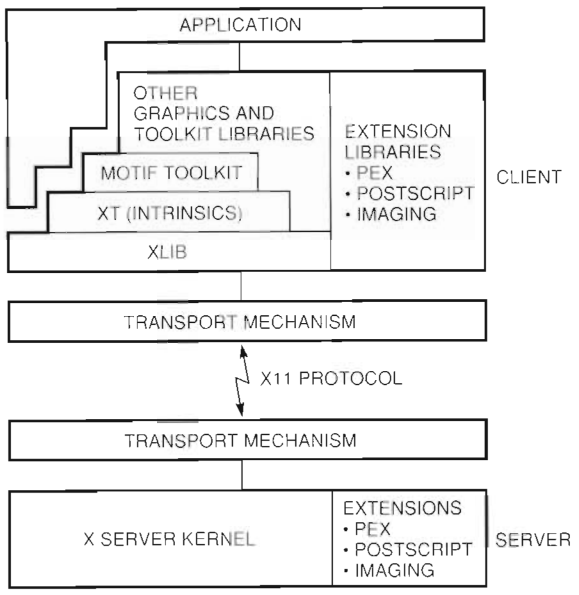

APPLICATION
OTHER GRAPHICS AND TOOLKIT LIBRARIES
MOTIF TOOLKIT
XT (INTRINSICS)
XLIB
EXTENSION LIBRARIES • PEX • POSTSCRIPT • IMAGING
CLIENT
TRANSPORT MECHANISM
X11 PROTOCOL
TRANSPORT MECHANISM
X SERVER KERNEL
EXTENSIONS • PEX • POSTSCRIPT • IMAGING
SERVER

Figure 2 X Client-Server Architecture

operating system, network transport, and network wiring technology and topology. The display server provides basic windowing, graphics rendering, and user input services for X applications.

eXcursion Implementation

The eXcursion application implements the X Window System display server on Microsoft Windows. The eXcursion software allows the windows of the X applications, running on a remote host, to display on a personal computer. The two environments are visually integrated—applications from both environments display on a single screen and have the same visual appearance. The two environments use the same mechanisms to manage windows and thus present a consistent user interface. In addition, eXcursion uses metaphors and mechanisms familiar to the user of Windows. A control panel is employed to handle configuration and customization of the eXcursion application. The Windows Program Manager is employed to transparently invoke applications on remote hosts.

Figure 3 shows the eXcursion control panel, the Windows calendar, and the DECwindows Motif calendar as viewed on a desktop device. The Windows Program Manager is also displayed to show the eXcursion program group with icons installed. Users can simply double click the icons in the program group to start applications on a remote host.

eXcursion—A Component of PC Integration

One of the goals of Digital's PC integration program is to integrate PCs throughout a network so they may share resources. In a local area network (LAN) or a wide area network (WAN), PCs share files and printers through a file server. Traditionally, Digital has provided terminal emulation software for interaction with a time-sharing system on the network. The X Window System distributes another resource load throughout the network, namely application services. X applications can be run on a special-purpose host, such as a CRAY system, or on a general-purpose host such as a VAX system. The applications share the CPU, memory, disk, and print resources of that host. Thus, the optimal or appropriate device can provide the application services. The eXcursion product is an X display server through which the PC user can access the X Window System class of application.

Because it enables information exchange among PC users and non-PC users throughout a network, the eXcursion software is a key component of Digital's Network Applications Support (NAS) and Digital's PC integration program in the Personal Computing Systems Group.

Design Philosophy

As in any software development project, a number of very important design goals and decisions were established for the eXcursion for Windows product which affected the implementation. The eXcursion application had to be extremely compatible with the Microsoft Windows environment. There were important reasons for this decision.

First, it was critical that eXcursion run on any PC, with any combination of devices that the standard Microsoft Windows environment supports. Typically, the manufacturer that builds the hardware is responsible for writing the Windows-compatible drivers. The devices that most affect eXcursion are keyboard, pointing device, and display.

Second, a tremendous amount of development effort has been invested in the functionality and performance of the Windows product. We wanted to apply that functionality and not duplicate it in the X server. For example, Windows software has a bit block transfer (BitBlt) routine that can more effectively handle that operation than eXcursion. It is one of the operations that Microsoft has optimized. In addition, it is one of the operations that

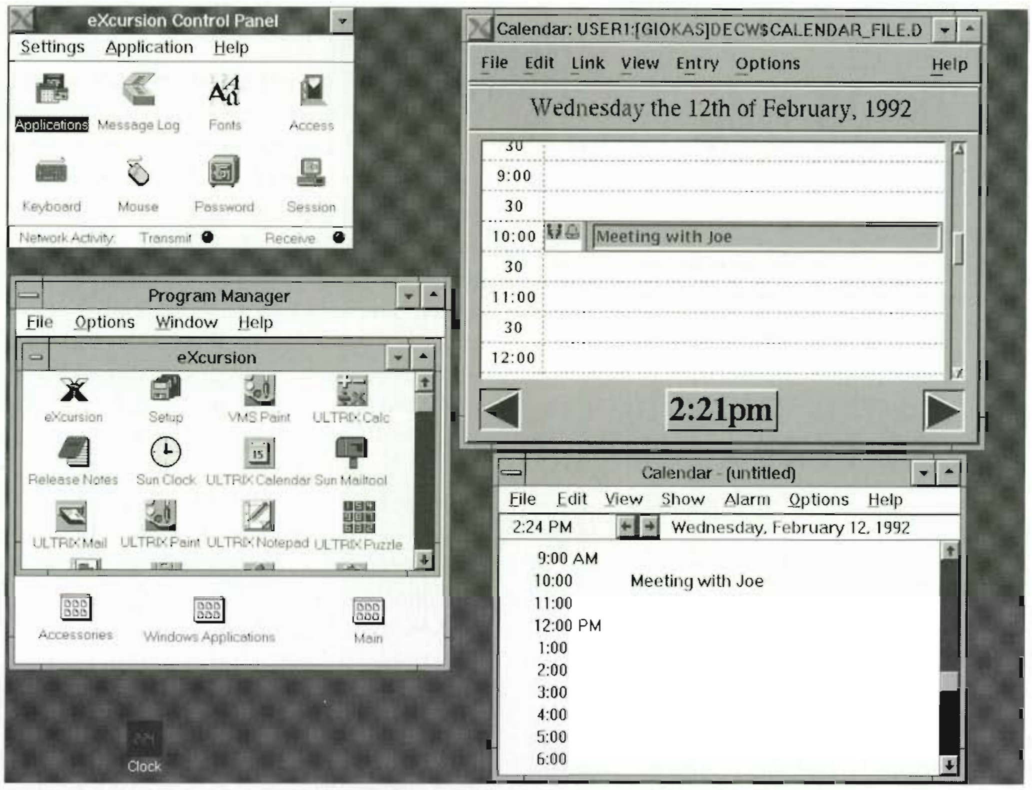

eXcursion Control Panel
Settings Application Help
Applications Message Log Fonts Access
Keyboard Mouse Password Session
Network Activity: Transmit ● Receive ●
Program Manager
File Options Window Help
eXcursion
eXcursion Setup VMS Paint ULTRIX Calc
Release Notes Sun Clock ULTRIX Calendar Sun Mailtool
ULTRIX Mail ULTRIX Paint ULTRIX Notepad ULTRIX Puzzle
Accessories Windows Applications Main
Clock
Calendar: USER1:[GIOKAS]DECW$CALENDAR_FILE.D
File Edit Link View Entry Options Help
Wednesday the 12th of February, 1992
30
9:00
30
10:00 Meeting with Joe
30
11:00
30
12:00
2:21pm
Calendar - (untitled)
File Edit View Show Alarm Options Help
2:24 PM Wednesday, February 12, 1992
9:00 AM
10:00 Meeting with Joe
11:00
12:00 PM
1:00
2:00
3:00
4:00
5:00
6:00

Figure 3 Windows Display with eXcursion

can be customized and optimized for the PC's graphics adapter. If the graphics adapter can handle BitBlt operations with built-in hardware, it is more likely that the operation can be performed faster with that hardware than with the CPU. Therefore, eXcursion is completely insulated from the hardware and benefits from functions that have been optimized for specialized hardware.

The third reason for developing eXcursion as a well-behaved Windows application is independence from the internals of the underlying windowing system. We might have been able to do a slightly better job of integration of the Microsoft Windows and X Window System environments if we had obtained a source code license from Microsoft and truly blended the two environments into one. However, the cost, development resources, and time needed to implement this type of integration were prohibitive.

Fourth, the eXcursion application had to share the PC system resources of display, pointing device (mouse), keyboard, sound subsystem, memory, and network with another windowing system and its applications. The first five resources were all owned and managed by Microsoft Windows. We had to use its application programming interfaces (APIs) to correctly share those resources. The network resource was shared among many networked applications through its APIs as well.

Use of Windows Resources

A substantial portion of the design debate centered on the way eXcursion would use the Microsoft Windows resources. We needed to determine how to map the windows, graphics contexts, fonts, and color maps of the X environment to the windows, device contexts, fonts, and color maps of the Microsoft Windows environment.

The major dilemma was: Should each X window be created as a Microsoft Windows window and thus be known to both environments? Or should only the top-level X windows—those which were parented by the Windows desktop or root window—be created as windows in the Microsoft

Windows environment, with all other windows created strictly as X windows and known only to eXcursion?

The first proposal was certainly easy to implement and it led to consistency throughout the X server. The Windows environment had an API rich enough to make this plan feasible. In addition, Windows would handle all the window stacking and clipping for eXcursion fairly transparently. Despite these reasons, the alternative plan was proven more workable during our prototyping phase.

The X Window System was designed to employ many windows since they are considered to be inexpensive resources.[1] Servers use little memory for each window. X windows are fast to create, map, unmap, and destroy; and they can navigate quickly through the window tree. Thus, X-based toolkits, such as Motif, employ many windows. When we tested our initial proposal, we discovered that both windowing systems maintained window trees, which resulted in a performance problem. For example, when certain operations such as graphics were performed, some of the clipping was done twice, once by eXcursion and once by Microsoft Windows. In addition, Microsoft Windows limited the number of windows that could be created, by the 64 kilobyte (KB) memory it reserved for these and other system resources.

Functionally, the X Window System graphics contexts (GCs) mapped fairly well to the Microsoft Windows device contexts (DCs). However, the way X applications employ GCs is significantly different from the way Microsoft Windows employs DCs. X applications store many GCs; each is set up uniquely with different values for the drawing state variables. Sometimes many GCs are used for one window and often a different GC is used for each window. The use of many GCs can significantly reduce the communication between the X server and application, since graphics state is communicated only once. Microsoft Windows applications use one DC for all window painting, modifying it as needed. Some innovative caching algorithms in the eXcursion product were used to address this mismatch in usage style.

Font resources were also efficiently mapped between the two windowing environments. A substantial portion of the graphics done by an application in a windowing environment is text. Microsoft recognizes this and optimized the text output routines in Windows. Thus, the optimal way of drawing text was through Windows. Therefore, the X server's font resources were compiled into Windows-compatible font file resources so Windows could do all the text drawing. For each X font resource, we included a second file for the font and glyph metrics that did not map to the Windows font file resource. Some of the eXcursion font file resources were modified to resolve inconsistencies between the two environments and make eXcursion compatible with Windows. For example, unlike X, Windows does not allow text drawing outside the characters' bounding box.

Color maps are another resource Windows shares with eXcursion. Microsoft Windows version 3.0 with standard video graphics array (VGA) hardware (a 640 by 480 resolution device with 16 colors supported) pre-allocates all 16 colors in the color table for the Windows environment. For eXcursion, this is effectively the X Window System static color visual, where the color map is read-only. With enhanced VGA cards that support 256 simultaneous colors, Windows pre-allocates 20 entries in the color table. For eXcursion, the X Window System's pseudocolor visual can be supported with only 236 entries for allocation in the color table. Again, it was important that eXcursion was well behaved with respect to color-map allocation and use within the Windows environment.

Performance Considerations

Performance of the eXcursion product is a continuing area of concern, investigation, and development. Many performance concerns were remedied by efficient code paths and innovative algorithms; others need to be addressed by the user in the form of trade-offs. In this section we discuss some major architectural differences between Microsoft Windows and the X Window System that leave X performance at a disadvantage when it is layered on another windowing system.

First and foremost, eXcursion has to translate X requests into Windows APIs as well as translate Windows events, API return values, and API errors into X events, X request replies, and X request error events, respectively. The disadvantage, of course, is the increased processing time eXcursion needs to complete these translation tasks. Since our design goal was to layer a foreign window system on the desktop device's native windowing system, we had to accept this performance penalty.

Second, X employs a client-server model. All X protocol requests of the X client (X application) to the X display server have to be encoded into the X protocol and transmitted to the server through an interprocess communication mechanism. For the eXcursion product, this mechanism is a network because the client and server are always on different systems. Operations in X, e.g., menu sweeping and resizing of objects, always involve both the client and the server. These operations in particular have to be fast because they affect the user's perception of the windowing system's performance. Thus these code paths had to be efficient.

Third, X has strict pixelization rules. These rules determine which pixels must be included in the rendering of a graphics object. In general, all the interior points of an object are rendered, but only certain points on the outer boundary of the object are rendered. If the area of the pixel below and to the right of the center point is touched, then the pixel is included; otherwise it is not.[2] Thus, a rectangle has its top and left edges included, but not its right and bottom edges. The pixelization rules for the X protocol were strictly specified to satisfy the technical market's graphics requirements, such as CAD/CAM. If one were to tessellate polygons in the X environment, one would be guaranteed that each pixel is included once and only once.

The Microsoft Windows environment was designed with a business graphics presentation model. The pixelization rules are not widely known and may change.

Based on these facts, we chose to adhere to the X protocol and its pixelization rules. We believed most users would run office productivity applications. For these applications, pixelization rules do not affect the operation or functionality of the application. In a majority of cases, the user is never able to see the subtle differences in the rendering of a graphics object. As part of eXcursion's customization, we allow the user to select the way graphics are rendered—optimized for performance or optimized for correctness. This choice is analogous to printing draft (fast) mode for proof copies or letter-quality, high-resolution mode (high quality but slow speed) for final copy. The user can change this parameter at any time in eXcursion and force a redraw by the X application, e.g., through an iconify/deiconify procedure, to render the graphics in the other mode.

Seamless Integration

One of our design goals was the seamless integration of eXcursion into the Microsoft Windows environment to the greatest extent possible. Two important areas to integrate were window management and data exchange.

Window Management We believed that Microsoft Windows should provide window management. Top-level windows in the two environments are peers and should be visually and functionally identical. With this capability the user does not have to run a remote window manager or learn and remember a second user interface.

We wanted the outer frame of the windows in X to look like the windows in Microsoft Windows. Furthermore, we wanted Windows to provide all of the end-user window management functionality—move, resize, iconify, deiconify, stacking, and focus. The windows for these operations had to contain the same user interface objects found in the Microsoft Windows environment. We did violate this design principle in one case. In place of the standard Microsoft Windows system menu icon in the upper left corner of the window frame, we placed an "X" (see Figure 3). This object visually cued the user that the window represented an X Window System application running remotely but displaying within the Microsoft Windows environment.

On the other hand, X servers are not aware if the graphics object being rendered is a component of a scroll bar, command button, radio button, check box, text entry field, etc. For this reason, eXcursion cannot make graphics objects look like and function as the equivalent objects in the Microsoft Windows environment. Unfortunately, the user has to deal with these inconsistencies between the two windowing environments.

The eXcursion product had to conform to the X Consortium's Inter-Client Communications Conventions Manual (ICCCM) specification for window management within the Windows environment. Window properties such as name, icon name, size, and position on a top-level window must be recognized by eXcursion and must be set using the appropriate Microsoft Windows APIs.[3]

Data Exchange We believed users should be able to seamlessly exchange text and bit-map data between the Microsoft Windows and X Window System environments. For example, the user should be able to use the standard application mechanisms to select data and cut or copy it from one environment, move to an application in the other environment, and use the standard application mechanisms to paste the data. No special user intervention between these two operations would be acceptable.

To enhance the data integration capabilities of eXcursion, we did implement a special feature to capture any part of an X window as bit-map data and save it in the Microsoft Windows clipboard. Microsoft Windows applications could then paste that data.

Cross-cultural Compatibility

eXcursion functions as any other Microsoft Windows application and conforms to its style guide in three areas—installation, configuration, and help.

The installation design principles are quite simple. Installation has to be performed through a Microsoft Windows application and has to allow the user to run the initial application without further configuration. Only two configuration parameters, fonts and keyboard, must be specified by the user. In addition, a user in the VMS, ULTRIX, or Sun OpenWindows environment has easy access to the standard applications of the operating system. The installation procedure installs icons that represent all of the standard DECwindows applications for the VMS and ULTRIX systems and standard Sun OpenWindows applications in the Microsoft Windows Program Manager. A user can invoke the application on the remote host using the standard Program Manager mechanisms, such as a double click of the program icon with the pointing device.

We devoted significant engineering resources to the configuration for eXcursion. Since the configuration was for a windowing environment, we decided to use the control panel metaphor that is common to other windowing environments, such as the Macintosh and Microsoft Windows. The eXcursion control panel (partially shown in Figure 3), provides access to all the user preference features and configuration parameters. Another important design principle was the immediate activation of configuration parameters or user preference features whenever it was technically feasible. We did not want the user to exit all the X applications or restart the X server to activate configuration parameters.

The eXcursion control panel also allows users to customize their X application environments. The eXcursion control panel provides a mechanism to build an applications menu within the control panel and install application start-up commands in the Microsoft Windows Program Manager as icons for easy invocation of remote applications.

On-line help also conforms to the Windows style guide. Our design goal was to supply a concise Quick Start card with all the information a user needed to determine the prerequisites for install, install the product, and invoke the first application. All of the remaining end-user documentation is available on line. The only other printed documentation is the reference manual.

For install, configuration, and help, human factors engineers provided usability evaluations, and a graphics designer assisted in the final design of the user interface.

X Server Internal Architecture

The X11 release 4 MIT sample server implementation provided the baseline for our development effort. This architecture is depicted in Figure 4. The sample server architecture has three distinct layers: device-independent X (DIX), operating system (OS), and device-dependent X (DDX). The DIX layer is primarily concerned with high-level decision making. The OS layer connects the X server to its underlying network transport. The DDX layer translates a client's request into a pixel display. To conform to

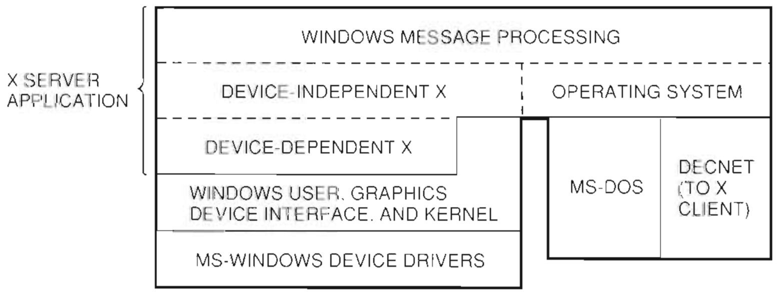

X SERVER APPLICATION
WINDOWS MESSAGE PROCESSING
DEVICE-INDEPENDENT X
OPERATING SYSTEM
DEVICE-DEPENDENT X
WINDOWS USER, GRAPHICS DEVICE INTERFACE, AND KERNEL
MS-WINDOWS DEVICE DRIVERS
MS-DOS
DECNET (TO X CLIENT)

Figure 4 eXcursion X Server Internal Architecture the Windows application model, our implementation adds a fourth layer, the Windows message processing layer.

Device-independent X

The DIX layer consists of modules that provide high-level server data structure manipulation, X request vectoring, and server task scheduling. Every attempt was made during the development process to change as little as possible in this layer, and to maintain the firewall between the DIX layer and the underlying DDX layer. The DIX layer's most important task is the dispatch loop, the scheduler for eXcursion processing of all asynchronous client requests. Requests fall into three categories:

1. Edits to internal data structures such as the current procedure vector for drawing wide, dashed lines
2. Queries on internal resources such as available fonts and their metrics
3. Drawing requests such as rendering of text and lines

The DIX layer maintains the current state of the window tree and all its components, as well as the graphics contexts and all of their associated data. DIX code dynamically alters the processing paths chosen for X request completion based on the current states of these data structures. For example, suppose that a GC is being used to draw a series of single-width, solid lines in a window. Now the X client wishes to begin drawing with 10-pixel-wide, tile-filled lines. DIX then reads the client requests dealing with the GC state changes, and updates its data to reflect the new drawing conditions for lines. DIX changes the drawing vector and updates the GC data structure. (Device-specific drawing operations are performed in the DDX layer.)

Windows Message Processing

The Windows message processing layer is the interface to the user's input devices, the mouse and keyboard. Actions taken by a user result in Windows messages containing information on the message type, conditions, and parameters being sent to the application's Windows message procedure. Here the data must be modified and translated into something that an X client can understand, an X event. Event processing is done by the DIX layer, and the event data is then shipped to the client by the OS layer.

Operating System

Data transferred on the X wire is arbitrated in the OS layer. When an X client application makes a server request, the underlying network code receives it, packages it, and makes it available to the OS layer. The eXcursion product runs layered above one of two entirely distinct network transports (either the DECnet or the TCP/IP protocol) and must provide some mechanism for passing data back and forth between the real mode of the network interface and the protected mode of a Windows application. For this reason, we chose to interface the server to the network by means of a generic OS module. Since all server-generated calls are now network-independent, the server is freed from any network-specific decisions.

Data conversions from real mode to protected mode are provided by a group of Windows dynamic link libraries (DLLs). Functions in DLLs are called directly from a Windows application (in this case, eXcursion). The DLLs in turn use Windows' extended memory manager to make DOS protected mode interface (DPMI) calls to pass the data to the network stack which runs in real mode. For example, assume eXcursion is running the TCP/IP protocol, and the user presses a mouse button in an eXcursion window. The data comprising the X event is assembled, packaged, and presented to the OS layer for shipment to the remote X client. The server makes its "send data" call into the generic OS module. This module makes a call into a common, shared DLL, and passes the data unchanged. The generic DLL acts as the network arbitrator. It knows about the underlying network transport and vendor since it performed a network installation check at start-up. Therefore, the generic DLL calls into the vendor-specific eXcursion DLL to modify the data, pack it into the format required by the network stack, and ship it to the real mode stack.

This implementation strategy requires several DLLs, but it completely shields the server, and more importantly the user, from the underlying network. The DLLs are simply copied once into the eXcursion execution path and forgotten. There is no need to reconfigure eXcursion if the underlying network changes.

Device-dependent X

All the visually recognizable work takes place in the DDX layer. DDX translates a client's X request into pixel manipulation on the screen. The sample server implementation that provided our starting point came with a DDX layer designed for monochrome frame buffer (MFB) devices. We replaced the MFB device-specific code in the DDX layer with implementation-specific code for Windows.

Our baseline sample server implementation also provided a machine-independent DDX mechanism (MI). The MI modules manipulate the video terminal as a virtual device: video memory is emulated and all drawing operations take place into this virtual space until the final output renders the bits onto the screen. The MI manipulates bits and performs logical operations until it achieves a final representation of the requested operation. This final drawing requires two distinct functions: fill spans and push pixels. The fill spans function renders drawing output in single scan lines, making repeated calls to Windows BitBlt. The push pixels function does much the same thing, but at a more complex level—it pushes bits through a mask or filter before they appear on the screen. These mechanisms are required for proper text rendition when tile or stippled filled text characters are requested with unaltered character outlines and backgrounds. These mechanisms are, by definition, clumsy and inefficient, but they provide pixel perfect renditions. eXcursion uses these MI functions when any of the following conditions must be met.

1. Drawings are complex filled areas.
2. Tile and stipples used are not 8 by 8 pixels in size. (Windows is optimized to handle this one case, and breaks down easily for all other sizes.)
3. All operations require pixel perfection, such as display of a CAD application.

Using Windows APIs

We designed a set of Windows-specific modules that filled the hardware-dependent space provided by MFB. These functions are called by the DIX layer's request dispatcher through the request vectors set up in the server's main data structures (screen, window, GC; see Figure 5 for examples). All X relative drawing requests are translated here into Windows operations, and Windows APIs are called to satisfy them.

As described previously, we decided to match window trees by creating a Windows window for each top-level X window only. X child windows are handled as if they are rectangular areas of their parents, thereby saving room in the finite (64KB total size) pool of Windows resources available for other objects. This decision led to a difficult problem that needed a solution: How do we handle window clipping?

Window Clipping

Clipping is accomplished in X by maintaining a list, for each window in the system, of the rectangles into which drawing is allowed. Clipping in Windows is accomplished essentially the same way, but it requires allocation of another resource, a region. We implemented clipping by adhering to the X model, letting the server code do as much of the work as possible.

The DIX code manipulates and maintains a "clip list" for each X window. When a Windows window is created and used, Windows expects this clipping information to reside in the window's DC if something is to be drawn in the window. To get the X clip list into the Windows DC, we allocated a small pool

```
if (gc.lineWidth == 0) {
  switch (gc.lineStyle) {
  {
    case Solid:           gc.line = GPXZeroLineSolid;
                          break;
    case OnOffDash:       gc.line = GPXZeroLineDashed;
                          break;
  }
}
else
  switch (gc.lineStyle) {
  {
    case Solid:           gc.line = GPXWideLineSolid;
                          break;
    case OnOffDash:       gc.line = GPXWideLineDashed;
                          break;
  }
```

Figure 5 Modifying Data Structures to Change Drawing Algorithms

of cached Windows regions. A DC (and X parallel GC) used for a drawing operation must be validated to ensure that all components are up-to-date. If the DC does not have a copy of the clip list, a Windows region is built from the rectangles in the X clip list and installed as the clipping region of the DC. When the drawing takes place, the clip list is installed. As long as the window is not moved, resized, or obscured, the region remains unchanged and further region validation is unnecessary. When the number of visible windows exceeds the cache limits, the least recently used DC is "thrown out" of the cache, and must be revalidated if it is used again. This mechanism allows smooth, efficient output to multiple windows without extensive use of Windows precious region resources.

Windows places a further restriction on resource usage. In addition to being created, a resource must be selected into a DC before it can be used. Deselected, old resources are deleted to save space. If a request asks for one of the deleted resources, it must be re-created and selected again. The caching and updating of DCs in Windows is handled by the same function that validates and refreshes GCs in X. When an X request results in a GC change, it may also result in a DC change. For example, if the line drawing mode changes from single-pixel-wide, solid fill to multiple-pixel-wide, tile fill, the GC is updated with new procedure vectors and data fields. At the same time, the DC must be updated so the next line drawing request results in a wide, tile-filled line. A Windows bit map is created for the X tile, and it is selected into the DC as the pattern. Any line then drawn using the DC results in a wide, tile fill. This method is used to update the DC whenever any GC object with a parallel Windows object is changed. The cache ensures that Windows objects can be allocated.

Drawing APIs

The Windows environment contains a rich collection of APIs designed to accomplish many types of drawing. The eXcursion application takes full advantage of these drawing APIs. Wherever X and Windows share drawing rules and conditions, the appropriate Windows API is called quickly to maximize performance. This mechanism is utilized when the user selects the "optimized for performance" drawing mode. When the rules between X and Windows differ, eXcursion calls the most appropriate API for the more common variants, again, to maximize performance. For example, since a wide, solid, horizontal line is rectangular, eXcursion calls the Windows FillRect API to draw it. Only rarely is the MI code path required.

Pixmap Manipulation

The X pixmap presented us with a major challenge. Since it is a bitwise representation of a visual object, its bit values must be maintained regardless of its use. Pixmaps can be used in a variety of ways by complex X client applications. Pixmaps can hold off-screen copies of window contents, or they can hold a pattern for a window background. They can provide a mask through which a color or pattern can be squeezed to give a stencil-like filling effect. They can also contain text characters prior to output.

The real challenge, however, lies in how pixmaps are manipulated. There are monochrome pixmaps, color pixmaps, pixmaps presented as an array of bits one color plane at a time, or packed to present each color plane for one pixel in succession. For these myriad forms and presentations we created a set of pixmap manipulation routines that translate back and forth between X and Windows. Since Windows provides a set of APIs for manipulating device-independent bit maps (DIBs), we stored the bit map internally in one, generic form regardless of its X representation. eXcursion extracts the bits, modifies them, and sends them to the client when it requests them in another format. One of the biggest performance bottlenecks in eXcursion lies in the pixmap format conversions which are constantly taking place under the surface. Since we have stored all pixmaps in device-independent format, the performance penalty is low.

Font Compiler

The X and Windows environments include a section dedicated to information about the font metrics and a section for the character bit maps. However, their font storage methods are different. Furthermore, since eXcursion is a compatible Windows application, it uses Windows fonts to draw text.

We designed a font compiler to create Windows-usable fonts from an X font file input. The font compiler takes a bit-map distribution format (.BDF) (X Window System font files are supplied in this ASCII readable format) and produces two output files. One, called the X font file (.XFN), contains the X metrics readable by the server without having to load the character bit maps themselves. The other, a Windows font file (.FON), contains the character glyphs used by the Windows APIs. eXcursion's X-specific code uses the .XFN file to match available fonts with those requested, and to calculate string sizes, positions, character offsets, ascents, descents, and anything else related to the location and position of the characters. The .FON file is loaded as a Windows resource, selected into a DC as described above, and used for any drawing operations since it contains the actual character representations. The font compiler can generate custom fonts; any font compiled with it produces a Windows font file suitable for use in any other, non-X, Windows application. For example, any of the supplied eXcursion fonts could be used with Word for Windows.

Handling Input Devices

In the section Seamless Integration, we described our design strategy for eXcursion to handle drawing requests from X clients. In this section we discuss requests from the user.

When a user clicks a mouse button, or moves the mouse, or types on the keyboard, Windows generates messages which are shipped to eXcursion's Window message processing function. Interrupt processing is not needed since Windows shields eXcursion from the underlying hardware. In fact, eXcursion has generic input handlers that work with any hardware configuration supported by Windows.

The message processor translates the data into a format understood by X, then packages and transmits it over the X wire as an X event. Since these user-initiated actions are asynchronous events, eXcursion calls the Windows PeekMessage( ) function when it has finished processing an X request, or when it is in the idle loop.

Windows and X share the same coordinate mapping conventions. When eXcursion receives a mouse move message, it does not perform translations on the x and y coordinates; it merely reports in which window the pointer resides. Furthermore, when eXcursion creates a window in Windows, it stores the corresponding X window's handle in the extra data area of the Windows window structure. It can retrieve the handle of a matching X window at any time with the Windows API GetWindowLong( ). Since eXcursion always matches a Windows window to a top-level X window, the combination of the top-level window handle and the x and y coordinates of the pointer allows eXcursion to scan the X window tree and determine which child window holds the pointer.

When a user presses a mouse button, the same kind of activity is used to determine which window contains the pointer. The X event data structure is filled in and shipped to the client for further action.

When a user presses a key on the keyboard, much the same processing takes place. Windows sends eXcursion all the information needed to build an event data structure containing the key state, the scan code of the key, and the key modifier state (whether Alt, Ctrl, or Shift is depressed). eXcursion then packages and ships the data structure to the client application.

eXcursion loads a keysym file at start-up. The file contains the keyboard mapping of hardware scan codes to keysym definitions for the user's keyboard. It permits custom configuration for a user's keyboard. The keysym compiler in eXcursion takes an ASCII text, keyboard mapping file as its input, and produces a binary keysym file as its output. As long as the user follows the layout of the input ASCII file, any key can be remapped in any way desired.

Manipulating Application Windows

As stated previously, eXcursion uses the Microsoft Windows window manager to manage and manipulate windows. Whenever the user moves, resizes, iconifies, maximizes, or closes a window, either by the Windows system menu or the mouse, Windows sends the eXcursion window procedure a message with specific parameters. For example, a message sent when a window is resized contains the old and new sizes and origins of the window. eXcursion translates every Windows input message into an X event and sends it to the X client.

Individual messages from Windows generally correspond to X event types that provide data to clients. However, complications arise when Windows generates multiple messages for a single action. For example, when a user presses a button to select an item from a menu, a new window is created, mapped, sized, placed on the screen, activated, and given the input focus—all as a result of the single user action. Windows messages are generated for each of these operations, yet the user has provided no further action.

To handle this extremely complex web, we benefited from our initial design decision to create only top-level Windows. We eliminated literally hundreds of Windows messages for each child window, simply by not creating them. Messages are sent only to the top-level window, and eXcursion can quickly determine which child (if any) needs attention. On the other hand, we had to observe and study window stacking, configuration, reparenting, activation, and window focus before we arrived at the final implementation. Only through extensive prototyping and empirical testing were we able to eliminate poor design choices and arrive at the best ones. As a result, every possible window manipulation action, whether initiated by the user or directed by a client, requires a translation from Windows to X and a careful selection of Windows function calls to keep the delicate balance between X and Windows.

Cutting and Pasting Data

To cut and paste data between X and Windows applications, we merged the Windows clipboard mechanism with the X selection mechanism by incorporating the cut/paste "pseudo-client" into eXcursion. This module watches for data cut-and-paste requests from X clients, as well as those from any Windows applications running on the PC. When it notices an X client gaining control of a selection, it asks the controlling client for the selected data, which it then puts into the Windows clipboard. The data thus becomes available to any Windows application with access to the clipboard. When a Windows application cuts or copies data into the Windows clipboard, the pseudo-client is notified, at which point it informs all X clients that it now owns the clipboard selection. X clients can then request the data from the pseudo-client by selecting paste from their edit menus.

Accessing Remote Applications

The user initiates remote X client applications through an application launching mechanism that provides several starting options.

1. Selection of an application from the eXcursion control panel's application pull-down menu
2. Selection from a dialog box of defined applications
3. Selection of the "Start X Application" dialog box
4. Double clicking on an icon installed for the application in the Windows Program Manager

The most interesting option, double clicking on an installed icon in the Windows Program Manager, allows the user to start up an X application without any knowledge of the current state of eXcursion. The double click activates XREMOTE.EXE, the remote application launcher. XREMOTE sends out a Windows message, with an identification known only to eXcursion. If eXcursion responds, XREMOTE passes it the command line for application start-up. If eXcursion does not respond within a short timeout period, XREMOTE issues a WinExec call, requesting start-up of eXcursion itself. Windows starts up eXcursion, passing it the command line string for the selected application start-up sequence. XREMOTE then terminates until the next start-up request.

Obviously, security is a major concern for any system that requires and handles account passwords; eXcursion application activation is no exception. Users log into their accounts by activating an X application such as DECterm. Two distinct passwords are required: (1) the eXcursion global, session password and (2) individual, application account password.

The eXcursion session password is optionally selected and set by the user from a control panel dialog box. It is stored as an encrypted string in the initialization file, and is used as the decryption key for the individual application account passwords, also stored in the initialization file. This design prevents an unauthorized person from using someone's .INI file to obtain access to an account. The user is prompted for the session password when eXcursion starts up. If an incorrect value is entered, the server terminates and application activation is impossible. A further level of security is provided by the "Prompt for Password" option, which the user can select for any application start-up.

Summary

The eXcursion for Windows display server seamlessly integrates the Microsoft Windows and X Window System environments. It provides a desktop integration tool that allows the user to display and interact with applications designed for both windowing systems at the same time. Data can be exchanged between them and desktop resources shared. A user is no longer required to work with two incompatible desktop devices in order to complete work assignments.

Acknowledgments

The authors would like to thank everyone who worked on the product during its development. In particular we would like to thank the other full- or part-time members of the software development team: Ray Shapiro, John Freitas, Mike Pfeffer, Lee Karge, Alice Chen, Mary VanLeeuwen, and Andy

Nourse. Two other members of the PC DECwindows Group who work on the DOS-based X server, John Wasser and Jim Peterson, provided some valuable assistance. We are also indebted to the following for their support and contributions: Emilie Schmidt, Carnel Hoover, Kathy Maxham, Andre Fontaine, Alice Chen, and Tracey Wemett. This great group of people made this project a joy to work on and a success.

References

1. R. Scheifler, J. Gettys, and R. Newman, *X Window System C Library and Protocol Reference* (Bedford, MA: Digital Press, 1988): xvii.
2. R. Scheifler, *X Window System Protocol* (Cambridge: MIT Laboratory for Computer Science, 1989): 37.
3. D. Rosenthal, *Inter-Client Communication Conventions Manual* (Cambridge: MIT Laboratory for Computer Science, 1989): 18-36.

General References

Digital Technical Journal, vol. 2, no. 3 (DECwindows Program, Summer 1990).

R. Scheifler, J. Gettys, and R. Newman, *X Window System C Library and Protocol Reference* (Bedford, MA: Digital Press, 1988).

Microsoft Windows Software Development Kit Reference, vols. 1 and 2 (Redmond, WA: Microsoft Corporation, 1990).

Microsoft Windows Software Development Kit Guide to Programming (Redmond, WA: Microsoft Corporation, 1990).

Christopher E. Methot

Capacity Modeling of PATHWORKS Client-Server Workloads

PATHWORKS network operating system software runs on the remote server computer that accesses files on behalf of clients connected to a network. The PATHWORKS file server provides clients with centralized backup, printing, and security. Popular desktop applications can be used in a manner that consumes large or small amounts of server resources. Capacity planning seeks to determine which network filing system is appropriate to current workloads and to predict capacity needs as the PATHWORKS client-server environment changes. The desktop industry lacks standardized performance tests. Digital has developed a general process that can be applied to any workload, including those in which the number of users causing the server process's resource consumption are unknown to a data collector. DECperformance Solution software was the primary tool used in the modeling process. Its analytical queuing model was used to predict performance and help define configuration alternatives.

The PATHWORKS network operating system software provides remote file service to desktop computing devices across a local area network (LAN). Integration of personal computers (PCs) on a network allows users to share applications, files, and printers. Most applications available on the desktop can be used in a manner that consumes widely varying amounts of that single-point resource known as the file server.

Some of this variation is due to the intentional part-time nature of the server's resource utilization, and some is caused by innocent changes in the user community's work techniques. Since desktop applications are used by novices and experts alike, small changes in the levels of skill, experience, and thus technique can significantly affect the performance of the server.

Capacity planning is a method of estimating the changing hardware needs for a computer system due to changes in workload. It can also be used to explore "what-if" alternatives for existing workloads.

Changes in user work habits such as running macros can increase a server computer's response time by as much as an order of magnitude. In addition, simplistic rules of estimating the consumption of server resources, such as number of users per VUP (VAX-11/780 unit of performance), can be very misleading. The use of applications in ways that increase individual productivity can slow server response time for the user community. These issues should be considered when selecting a file server system. Because the number of active users is often unknown in client-server environments and the user application technique may vary, capacity planning uses a model of the actual workload to predict server performance and help define configuration alternatives.

This paper describes a queuing analytical model that was used to gain knowledge about resource consumption on the PATHWORKS server computer. The paper discusses the special modeling process required for the client-server environment. It describes data capture and workload classification using DECperformance Solution software. Finally, the paper presents the results of a performance analysis of a PATHWORKS server with response-time constraints.

Some of the terms found in this paper have specific definitions. Many of the "correct" terms for network file serving are not the terms used by users of these systems. Network file serving has acquired the name "networked." Server computers are often referred to as "the network," and getting access to one's files on the server is usually called logging into "the network." In this paper, we refer to MS-DOS-based PCs and Macintosh computers generically as desktop computing devices. In addition, the word "workload" refers to the cause of the resource consumption, which is the combination of client application and user technique within that application. The term "workload class" has a specific definition in DECperformance Solution software. It refers to a group of VMS processes that the modeler wants to manipulate differently from other processes.

Defining the Question

PC users on an integrated PATHWORKS network need to determine which server computer system is appropriate to their workloads today, and which will be appropriate as their numbers increase in the future. The system they choose must deliver sufficient performance today and allow a method to plan for expanded needs in the future. Users of desktop computing devices, which are not networked, can benefit from a series of anecdotal model case studies which describe other workloads and the file servers which were recommended. This paper gives the results of our efforts to gain insight into the reasons for and symptoms of server resource exhaustion (bottlenecks) on PATHWORKS file server systems.

Analytical Models

PATHWORKS software takes advantage of the expanded computational power of the client-server architecture, which requires special modeling techniques. Two of Digital's analytical modeling tools can be used in our capacity modeling process, however, DECperformance Solution was the primary tool. The model was used to answer questions about the need to enhance file server computer resource requirements as a result of changes in hardware or workload.

Performance models can answer at least two questions.[1] First, "How is performance affected if we change either the number of users or the amount of hardware?" Second, "How can we maintain performance if we add users doing the same kinds of tasks?" Of the two, the second question is the one we seek to answer when we model PATHWORKS client-server workloads.

Data Collection

Data can be collected with the VAX Performance Advisor (VPA) version 2.1 or the DECperformance Solution version 1.0 or later. DECperformance Solution software is an integrated product set that provides performance and capacity management capabilities for computing systems. This layered software product runs on the VAX VMS operating system and uses a queuing analytical model to answer questions. This process requires collection of two kinds of information.

1. A detailed record of the cause of resource consumption, including which process is causing each disk or CPU activity. Processes should be combined into like groups, called workload classes, which may be manipulated independently. For example, some workload classes may be reduced or eliminated and some may be increased.

2. As detailed a record as possible of the effect of resource consumption, including the effect on multiple remote clients. Changes in performance are typically measured by the elapsed time from the carriage return to the return of the prompt. In the case of a timeshare user, this is a closed loop since almost the entire process is visible to the data collector.

In a PATHWORKS environment, such data capture is not possible. A data collection device running on the server computer cannot determine the number of users for whom the PATHWORKS server process is consuming resources. Furthermore, the collector cannot detect the response time seen by the users of the desktop devices.

We have developed a general process that can be applied to all client-server workloads. These include applications such as VTX or VAX Notes, in which the number of users initiating the server process' resource consumption are unknown to a data collector.

Figure 1 illustrates a simplified closed queuing model of a PATHWORKS transaction. The user initiates the transaction through a keyboard or pointing device. The application running on the desktop computer performs the initial local processing and issues a call to the server requesting I/O. The server performs some remote computing, and the I/O request is satisfied when the server transmits either the data or acknowledgment that the data has been written. This travels back to the user's desktop device and some further computing leads to a graphic indication to the user to proceed to the next step.

If these three sequential queues—client, network, and server computer—were equal in response time, the server would have only a one in three influence on the responsiveness the desktop user sees. Of course in reality the three queues are never equal, and the two local queues are highly dependent on the local desktop computer's capabilities. Each queue can have a request backlog if the service time is not faster than the arrival rate. The response time of any queue is the queue wait time plus the actual time to be serviced. The total response time of the workload class, as modeled on the server, is the analytic sum of all its queues' response times.

In reality, the analytical model of the PATHWORKS environment is more complex than the one shown in Figure 1 and involves disk, memory, and CPU queues. The response time calculated for a PATHWORKS server computer workload class is the calculated sum of the response times of all server process queues for that workload class. As stated earlier, this is only an indicator of a desktop user response time.

Cause and Effect

A data collector, running on the server computer is not aware of the response time perceived by the user at the desktop device, nor can the server's data collector process know how many users are generating the current workload. Server response time is a subset of the response time as seen at the desktop; and if the server's response time improves, the user's will improve as well, as shown in Figure 1.

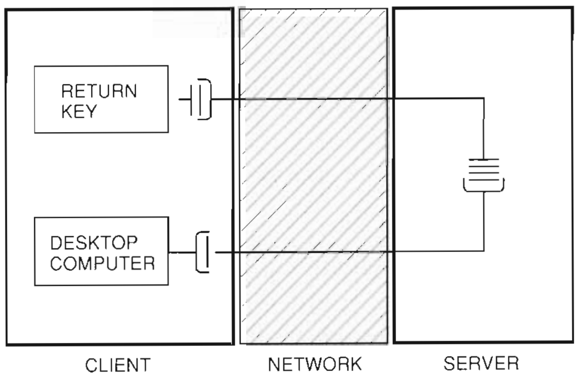


Figure 1 Simple PATHWORKS Queuing Model

A model that is built from a data collector which has only a partial definition of the whole loop (i.e., the server computer portion as shown in Figure 1) is called an open model.[2] The models described in this paper are open models. Since the most likely bottleneck is the shared resource known as the server, this is a useful way to model client-server workloads.

Uniform Service Level

Model analysis of a PATHWORKS client-server computer workload cannot predict the increase or decrease in response time seen by the user. A model can determine the effect of any change in hardware configuration or arrival rate (number of users). Capacity planners can use this method to add more users by incrementing arrival rates. Then hardware can be upgraded until an equal or faster server response time is reached. This method can be used to increase the number of users at the same performance or split users into smaller groups with the same or better performance.[1]

Not all desktop transactions require server intervention. In fact, the success of the client-server architecture depends on infrequent access to servers. Obviously, file servers are required when a file is saved. However, many applications perform disk I/O without any obvious or explicit user action. For example, WordPerfect software provides a temporary file that is a type of journal file. Periodically, the application updates this file with data stored in memory. When a user's input reaches a predefined buffer limit, the next keystroke causes the file to be written. The capabilities of this application, and many others, must be considered when planning the capacity of a PATHWORKS file server installation. In this example, the load per client on the server can be significantly reduced by placing the temporary file on a local hard disk.

Performance of a file server computer can also be affected when expert users employ macro techniques or when users generate automated output. Macros read each instruction from the macro file one record at a time, thereby continuously doing I/O. Most expert users provide a save as the last instruction in the macro, which allows them to be absent when the work is being accomplished and then saved. This increases server I/O as well. Most desktop applications permit automated output. For example, some allow form letter generation;

some computer-aided design (CAD) applications provide Bills of Materials. This capability also increases server I/O.

The use of either macro techniques or automated output can impact server computer utilization. A server that was intended to be a part-time file server can become a full-time I/O device which can rapidly exceed its capacity.

To illustrate how a small change in environment can affect file server performance, we employed a Markov model, using a SHARPE queuing model of a server environment. Figures 2 and 3 show the results. We asked the question "If we had 120 users each randomly filing once an hour and each file action took 5 seconds, how often would a user wait for another user to complete a file transaction?" We discovered that only 14 percent of the time another transaction would be running in the server process. Then we asked, "What would happen if 5 of the 120 users started running a macro and this macro did I/O for 5 minutes at random intervals within the hour?" The remaining 115 users continued working as before. In this case the possibility increased to 28 percent that a job request would be on the queue, 24 percent that two job requests were waiting, and 20 percent that three job requests were present.

In the same study, less than 5 percent of the users changed the way they were working. None of the applications was changed. Almost any PC or Macintosh application can reasonably be used in this way. As the smaller group of users became more productive, the other 95 percent experienced a significant delay in response time. The system capacity must be sized to allow for a situation in which user activity lessens overall response time.

Modeling Process

The modeling process we describe in this paper was developed over a two-year period. Before discussing the modeling procedures, we list the benefits and limitations of the process.

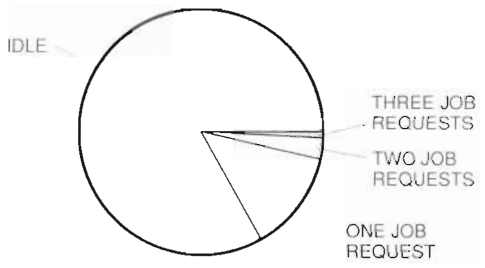


Figure 2 Low Use with Infrequent Saves

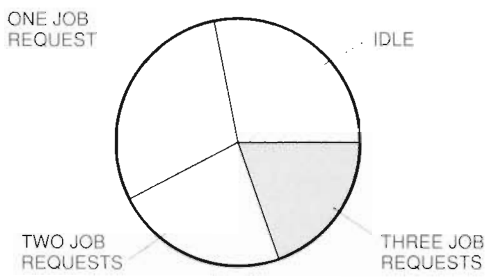


Figure 3 High Use with Few Macros Running

Benefits

- Determinations can be made as to the numbers of PATHWORKS and new workload class users required to maintain the same performance.
- Single-function server computer models, with only PATHWORKS workload classes, can have non-PATHWORKS workload classes added for a more complex environment.
- The server can be upgraded to maintain the performance level of growing user communities.
- Larger user communities can be divided between two standalone servers to maintain an acceptable level of performance.
- Stable user communities can be reduced to provide equal levels of performance with two smaller servers.
- Hardware trade-offs can be explored. For example, some users can be moved to another disk.
- Local site management can be made aware of the magnitude of daily workload variation; understanding this variation is also part of the model process.

Limitations

- The model cannot predict response time changes at the client, due to changes in server loading.
- Information about the number of users generating the applied workload must be collected by methods other than using DECperformance Solution software. These methods are detailed in the section Capturing Workloads.
- Although memory can be modeled, the model cannot anticipate the increased PATHWORKS

read or record management services (RMS) cache requirements. When adding users to a PATHWORKS server computer, adequate spare memory must be allowed to provide the same or better cache hit rates. The RMS cache hit rates can be determined, without software tools, by executing a program at the Digital command language (DCL) prompt: @SYS$UPDATE:AUTOGEN SAVPARAMS TESTFILES FEEDBACK, and then reading SYS$SYSTEM:AGEN$PARAMS.REPORT.

- Available modeling tools only allow PATHWORKS workloads to be modeled onto VAX VMS servers.
- Prior to data collection, the server must be checked to see if it is tuned for use today and for the future, or the recommended server system may be incorrectly sized.[1]

Capturing Workloads

DECperformance Solution software requires VAX Performance Advisor version 2.1 or later collector files named nodename_date.CPD. In addition, either a VPA$SCHEDULE.DAT or a PSDC$SCHEDULE.DAT file is required to define the cluster configuration and collection schedule. Either a VAX Performance Advisor version 2.1 or DECperformance Solution version 1.0 Data Collector, or the DECperformance Solution Service Delivery Software kit may be used to collect data. All three require a license and product authorization kit.

Enough data must be collected to represent the range of a typical workload. The sum of the subjective user opinion of performance must be collected as well as the tasks the users were performing. If this data is not collected, the planner may mistakenly model equal levels of user dissatisfaction rather than equal levels of user satisfaction. Subjective performance evaluation is always gathered by interviewing or monitoring users.

Collections should be made over a series of normal workdays to avoid gathering misleading data. We have observed two normal workdays with only a 5 percent difference in the number of desktop users logged into the server, yet five times more server resources were used.

Additional data on user activity that is consuming resources must be collected by methods other than the DECperformance Solution collector. Both the Macintosh and MS-DOS server products have interactive DCL software utilities that provide some information about the condition of the current server process. Command procedures can call these utilities with a brief DCL command string. For example, ADMIN/PC SHOW FILE COUNTERS displays the current cache misses and request rates, and ADMIN/PC SHOW FILE SESSIONS shows the client device ID, client connections, and open files. The size of the server process cache configuration can be gathered using the ADMIN/PC SHOW FILE CHARACTERISTICS command. If analysis is performed offsite, a DCL procedure can gather information about volumes and system logical names, which allows user disk assignments to be defined. Finally, user authorization resource limits on the server process can be extracted from the system. The Macintosh server software has similar commands using the ADMIN/MSA SHOW CONNECTION command.

When the size of the user community is unknown, the above data must be used to characterize the number of users being modeled. Specific customers with large installations or many remote sites need quantitative user characterization. In all cases the cause of the observed performance characteristics must be determined at some quantitative level.

The data gathered by using the ADMIN/PC SHOW FILE COUNTERS and ADMIN/PC SHOW FILE SESSIONS commands can be invalidated if desktop devices include automated procedures to attach to file services when the desktop device is booted. The simple act of activating the client power switch should not count that user as explicitly intending to use the server computer. On the other hand, explicitly connecting to file services and being interrupted for an unexpected event should not exclude that user from the total active user count. Ultimately, a combination of the total possible and the total active connections is needed.

Defining Workload Classes

With the DECperformance Solution data collector, workload classes are defined prior to starting the modeling process. They are defined either by specifying the anticipated logical divisions or by determining them from the observed performance data. DECperformance Solution software provides many ways to group processes, e.g., user identification code (UIC), resource usage, image name.[3]

The DECwindows interface to the performance tool DECperformance Solution provides an excellent way to review the data.[1] The graphic display of the server process by day along with the subjective user characterization can help select the day or days to be modeled. The same method can be used to determine peak usage hours. Finally, this technique can help categorize workload classes by applicable processes. Table 1 lists the workload class groupings we used.

Workload families are groups of workload classes that the data collector can expect to see. The PW_DOS workload family characterizes a system as a PATHWORKS file service environment. It includes PATHWORKS server processes, required system overhead functions, and processes needed to collect data that are not normally part of the system. All other processes are automatically placed in a category called "other." This suits the needs of our general-case, single-function PATHWORKS server computer, but any server can be used for tasks unrelated to the PATHWORKS print and file service. If the tasks in the default (other) category need to be subdivided for separate scaling, the workload class definitions have to be added to a family which calls each workload class explicitly, as indicated for the PW_LAD workload class family in Table 1.

For example, consider the question "As groups of ALL-IN-1 system users change to PCs, how many users can the PATHWORKS server computer support?" This determination requires defining another workload class by UIC for the ALL-IN-1 system users. The workload class could be moved by UIC to the FILESVS workload class. This method assumes the current collection of FILESVS workload classes reflects the mix of the remaining ALL-IN-1 system users.

Even before the model building step takes place, the PSDC$DATABASE logical must be pointing to the location of the VPA$SCHEDULE.DAT and the VPA$PARAMS.DAT files. The model building step generates a model with the workload class groupings given in Table 1. The workload class and family definitions are made using the DCL command ADVISE PLAN EDIT in the VPA/VME (VAX Performance Advisor/VAXcluster Modeling Environment) utility and are written to a file named VPA$PARAMS.DAT. (If the DECperformance Solution tool is used, the files are named PSDC$SCHEDULE.DAT and PSDC$PARAMS.DAT.)

If this logical is defined while using the DECperformance Solution DECwindows interface invoked from the session manager, the logical may not take effect in the DCL session in which the model is to be built. The command to generate a model can include the time selected to be representative and the workload class family definition name. A report can be generated which describes the newly built model. The command used is: ADVISE PLAN BUILD/CLASS=(USER=PW_DOS)/BEGIN=9-DEC-1991:10:30-/END=9-DEC-1991:11:30/REPORT/OUTPUT=MYMODEL.RPT MYMODEL.MDL.[3]

At this point the model must be validated by typing ADVISE PLAN REPORT MYMODEL.MDL VALIDATION/OUTPUT=MYMODEL_VALID.RPT at the DCL prompt. All predicted values should be within 10 percent of the calculated values.[2,3] A CPU validation report for a collected workload includes data on throughput, queue length, average service time, average response time, and percent of utilization. For the FILESVS workload, the measured utilization was 67.7 percent as compared to 64.7 percent for the model. This 3 percent difference is 4.4 percent of the measured value and thus well within the 10 percent range.

Normalizing the Environment

The next step is to return the system to the normal environment. Even though data collectors are typically designed to utilize a small amount of system resources, they are not normally part of the

Table 1 Workload Class Groupings

Workload Name	Image Name Selection Criteria
FILESVS	NETBIOS, PCFS_*, PCSA$*
OVERHEAD	AUDIT_SERVER, NETACP, EVL, ERRFMT, OPCOM, JOBCTL, REMACP, CONFIGURE, IPCACP, TPSERVER, FILESERV, CSP, SMISERVER
ABNORMAL	PSDC*, VPA$DC_V5, DECC*, SPM, MONITOR
MAC_FILESVS	ATK*, MSAP*, MSAD*, MSAF*
LAD	LAD$KERNEL
OTHER	(All Else)

Workload Family	Workload Member(s)
PW_DOS	FILESVS, OVERHEAD, ABNORMAL
PW_MAC	MAC_FILESVS, OVERHEAD, ABNORMAL
PW_BOTH	FILESVS, MAC_FILESVS, OVERHEAD, ABNORMAL
PW_LAD	LAD, FILESVS, OVERHEAD, ABNORMAL
PW_THREE	LAD, FILESVS, MAC_FILESVS, OVERHEAD, ABNORMAL

server workload. Grouping abnormal processes into a workload makes it easier to remove them during the DECperformance Solution model process. Access to the DECperformance Solution model interface is achieved through the command ADVISE PLAN MODEL MYMODEL.MDL.[3]

Recording Response Times

The next step is to solve the model and view the calculated response times for the remaining workload classes. These are FILESVS, OVERHEAD, OTHER, and any custom-defined classes. The OTHER workload class can be used as a defined workload class provided it contains no unexpected processes that are using significant resources. The calculated response times for the remaining workload classes should be considered maximum times, and model manipulations should always seek to attain these numbers or less.

If the intention is to capture the PATHWORKS workload class for use elsewhere and if the same system had significant OTHER workload classes, these classes should be removed (turning the server computer into a single-function PATHWORKS server).[3] This reduces the response times of the remaining workload classes and requires increasing the PATHWORKS workload class until the response time returns to the observed value. The increase in throughput is proportional to the increase in PATHWORKS users accommodated at the same performance, without the competition of the OTHER workload class.

Model Manipulation

Basically, the response time can be manipulated (1) by decreasing the usage of a significant resource (model resource utilization percentages help locate the bottlenecks) or (2) by increasing the capacity of that resource.

There are two ways of decreasing the resource utilization. If the resource is single-threaded on the critical path, as a CPU would be in a non-symmetrical multiprocessor (SMP) machine, the method is to reduce the number of users by decrementing their arrival rate (called throughput or transactions per second [TPS] in various menus) or by increasing the speed of the bottlenecked device.

The model allows for workload class manipulation to remove arrival rates of the workload class. As this is being done, the original arrival rate must be noted so the same changes can be applied to the number of users that caused the workload.

If the bottleneck is not on a single path, its capacity can be increased by spreading the load across another similar device. This can be achieved with multiple disks.

In the ALL-IN-1 system case discussed earlier, 100 percent of the workload class from the first UIC group of ALL-IN-1 system users can be removed from the model.[3] If the model is solved at this point, all the workload class's response times should diminish. If the FILESVS workload class throughput is incremented in proportion to the additional PATHWORKS users and the model is solved again, the response times of all workload classes increase.

The question is: "Has the removal of the ALL-IN-1 system users decreased critical resource usage sufficiently that their addition to the PATHWORKS FILESVS workload class does not increase any of the remaining workload class's response times beyond their target?" The answer depends on the per capita usage of the critical resource of each workload class. The nature of each workload class may be different. For example, PATHWORKS workloads do not scale well over SMP processors. The workload class being removed may use more CPU time per user than the PATHWORKS FILESVS workload class.

Findings

We analyzed a large PATHWORKS workload class from a VAX 6000 model 510 system whose CPU utilization averaged 72 percent. The subjective user evaluation was that this system was very near performance capacity limits, and a fair amount of dissatisfaction was associated with the level of performance. The question was asked "Could this community be split in half across two VAX 4000 model 300 systems with the same or better performance?" We immediately agreed this would work, but went about proving it with a model. After the workload class was normalized and the response times were noted, the workload class arrival rate was reduced by 50 percent and the CPU and disk systems were changed to the VAX 4000 model 300. The new model was solved, and the response times were significantly worse than with the VAX 6000 model 510 system. The workload class was halved again, and the resulting response time was still slightly over the target.

This finding was difficult to understand since the VAX 4000 model 300 system CPU was now down to 36 percent utilized, and only one quarter of the users remained. The reason for the inadequate response time was found by studying the queuing model. Figure 4 is a simplified model showing two CPUs and their queues displayed on a time scale. The first is a slower CPU and the second a faster one. Since we did not allow the response time (total queue plus service time) to vary, the queue length (measured in number of waiting jobs) on the slower CPU was shorter. The service time of the slower CPU was larger, in proportion to its queue wait time, and therefore an interruption by an overhead process caused significant loss of processing time (response time) to be available for the critical workload class.[5]

Therefore, the general rule became: Slower CPUs will be less utilized at the same workload class response time. This result has been seen on two different customers' workload classes (one with DOS and one with Macintosh clients) which were modeled by different engineers using different modeling tools.

Another surprising result became evident in the day-to-day variation at a customer's installation. The same two workload classes were analyzed across several days to examine typical workday variations in workload class resource utilization. Two normal workdays were selected by the customer. The most intense hours of these two days were different by a significant factor. On one workday, three to five times as many users applied the same workload class as on the other day, yet all experienced the same response times. This wide variation is typical of client-server workloads.

Library of Workload Classes

After we had captured a series of data, we created a small library of real workloads that represented various conditions. The actual workloads consist of a

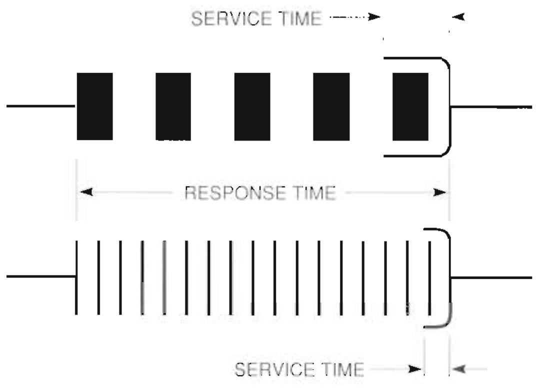


Figure 4 Server Queue Comparison on Different CPUs

model file that is devoid of user-specific information. Other non-PATHWORKS workloads can be added to these models. Alternatively, the numeric workload characterization can be added to existing models. Using the above methodology, the model can be manipulated to determine what system is appropriate for this more complex environment. As additional installations are analyzed, their model files will be added to the library.

With either the DECperformance or DEC Capacity Planner modeling tool, the process is the same: Change the hardware and modify the throughput to maintain or lower the response times of the model during iterations. The changes to throughput are then applied to the original number of users to determine the acceptable number of users in terms of server computer capacity.

Although both modeling tools exhibit similar mapping of the quantitative workload class characterization, we do not know the units of some of the key metrics used. Therefore, entering a workload class captured in one model to another model is not recommended.

Summary

The PATHWORKS network operating system software provides remote file service to desktop computing devices across a local area network. Capacity planning of client-server environments requires the use of special modeling techniques. DECperformance Solution software provides performance and capacity management capabilities for computing systems; it uses a queuing analytical model to answer resource consumption questions. The modeling process depends on the collection of enough data to represent the range of a typical workload. Additional data on user activity that consumes server resources must also be collected. Analysis of workload models reveals the reasons for and symptoms of bottlenecks. Capacity planning depends on the results of these analyses to predict server response times.

Acknowledgments

I would like to thank Prashant Bhabhalia, who helped me ensure that the modeling process is correct, and Dick Dunnington, who checked my queuing theory. Also, I would like to thank Frank Caccavale, who helped me understand the PATHWORKS server architecture. Melur Raghuraman helped me grope toward the Uniform Service Level model described here. Karl Friedrich, Ann Bousquet, and Lindsey Stephens helped me transition to DECperformance Solution software. Finally, I would like to thank Pete Stoddard for applying his technical reviewer skills to this paper.

References

1. *Guide to DECcp Methodology* (Maynard: Digital Equipment Corporation, Order No. AA-NA34A-TE, 1989).

2. R. Jain, *The Art of Computer Systems Performance Analysis* (New York: John Wiley & Sons, 1991).

3. *DECperformance Solution Capacity Planner User's Guide* (Maynard: Digital Equipment Corporation, Order No. AA-PH6LA-TK, August 1991).

4. *DECperformance Solution Performance Advisor User's Guide* (Maynard: Digital Equipment Corporation, Order No. AA-PH6SA-TK, August 1991).

5. F. Hiller and G. Lieberman, *Operations Research* (San Francisco: Holden Day, 1967).

Further Readings

The Digital Technical Journal *publishes papers that explore the technological foundations of Digital's major products. Each* Journal *focuses on at least one product area and presents a compilation of papers written by the engineers who developed the product. The content for the* Journal *is selected by the Journal Advisory Board. Digital engineers who would like to contribute a paper to the* Journal *should contact the editor at RDVAX::BLAKE.*

Topics covered in previous issues of the *Digital Technical Journal* are as follows:

Image Processing, Video Terminals, and Printer Technologies
Vol. 3, No. 4, Fall 1991

Availability in VAXcluster Systems/ Network Performance and Adapters
Vol. 3, No. 3, Summer 1991

Fiber Distributed Data Interface
Vol. 3, No. 2, Spring 1991

Transaction Processing, Databases, and Fault-tolerant Systems
Vol. 3, No. 1, Winter 1991

VAX 9000 Series
Vol. 2, No. 4, Fall 1990

DECwindows Program
Vol. 2, No. 3, Summer 1990

VAX 6000 Model 400 System
Vol. 2, No. 2, Spring 1990

Compound Document Architecture
Vol. 2, No. 1, Winter 1990

Distributed Systems
Vol. 1, No. 9, June 1989

Storage Technology
Vol. 1, No. 8, February 1989

CVAX-based Systems
Vol. 1, No. 7, August 1988

Software Productivity Tools
Vol. 1, No. 6, February 1988

VAXcluster Systems
Vol. 1, No. 5, September 1987

VAX 8800 Family
Vol. 1, No. 4, February 1987

Networking Products
Vol. 1, No. 3, September 1986

MicroVAX II System
Vol. 1, No. 2, March 1986

VAX 8600 Processor
Vol. 1, No. 1, August 1985

Subscriptions to the *Digital Technical Journal* are available on a yearly, prepaid basis. The subscription rate is $40.00 per year (four issues). Requests should be sent to Cathy Phillips, Digital Equipment Corporation, ML01-3/B68, 146 Main Street, Maynard, MA 01754, U.S.A. Subscriptions must be paid in U.S. dollars, and checks should be made payable to Digital Equipment Corporation.

Single copies and past issues of the *Digital Technical Journal* can be ordered from Digital Press at a cost of $16.00 per copy.

Technical Papers by Digital Authors

R. Al-Jarr, "Performance Modeling of Computer Systems: The Petri Net Approach," *Computer Measurement Group Conference* (December 1991).

S. Angebranndt, R. Drewry, and T. Newman, "Writing Tailorable Software: The X11 Sample Server," *Software* (October 1991).

P. Anick, "Lexicon Assisted Information Retrieval for the Help-Desk," *Eighth IEEE Conference on Artificial Intelligence Applications* (March 1992).

N. Arora and M. Sharma, "Modeling the Anomalous Threshold Voltage Behavior of Submicrometer MOSFETs," *IEEE Electron Device Letters* (February 1992).

S. Bazydola, "An Experimental Investigation of a Staggered Array of Heatsinks in the Hydrodynamic and Thermal Entrance Regions of a Duct," *I-THERM III* (February 1992).

D. Bhavsar, "An Architecture for Extending the IEEE Standard 1149.1 Test Access Port to System Backplanes," *IEEE International Test Conference* (October 1991).

E. Braginsky, "The X/Open DTP Effort," *Fourth International Workshop on High Performance Transaction Systems* (September 1991).

X. Cao, "An Introduction to Ensemble-Average Importance Sampling of Markov Chains," *Proceedings of the Thirtieth IEEE Conference on Decision and Control* (December 1991).

R. Cembrola, "Analytical Chemistry in Support of Microelectronics Technology," *Boston Section Meeting of the American Chemical Society* (November 1991).

Z. Cvetanovic and E. Freedman, "Efficient Decomposition and Performance of Parallel PDE, FFT, Monte Carlo Simulations, Simplex, and Sparse Solvers," *The Journal of Supercomputing*, vol. 5 (1991).

S. Denker, "A Common Sense Approach to Improving the Design and Management of Electronics Manufacturing Processes," *International Conference on Automated Materials Handling* (1990).

B. Doyle, R. O'Connor, K. Mistry, and G. Grula, "Comparison of Trench and LOCOS Isolation for Hot-Carrier Resistance," *IEEE Electron Device Letters* (December 1991).

B. Fishbein, D. Krakauer, and B. Doyle, "Measurement of Very Low Tunneling Current Density in SiO2 Using the Floating-Gate Technique," *IEEE Electron Device Letters* (December 1991).

W. Harris, H. Smith, and A. Pelillo, "SIMS Test Structures for Analyses of Semiconductor Product Wafers," *American Vacuum Society Thirty-eighth National Symposium* (November 1991).

D. Heimann and W. Clark, "Process-Related Reliability-Growth Modeling—How & Why," *IEEE Reliability and Maintainability Symposium* (January 1992).

S. Heng, H. Pei, and J. Watson, "Closed-Loop Cooling for Computers—Opportunities for the 90s," *National Electronic Packaging and Production Conference* (June 1991).

S. Knecht, "Integrated Matrix Creep: Application to Lifetime Prediction of Eutectic PbSn Solder Joints," *Materials Research Society Symposium Proceedings* (November 1990).

L. Lee and B. Mirman, "Bonding Quality and Bending Stiffness," *International Electronics Packaging Society Conference* (September 1991).

M. Lefebvre, "Test Generation: A Boundary Scan Implementation for Module Interconnect Testing," *IEEE International Test Conference* (December 1991).

R. Jain, "The Art of Computer Systems Performance Analysis," *Computer Measurement Group Conference* (December 1991).

J. McGrath and J. Derosa, "3-D Solid Modeling for IC Assembly," *IEEE Advanced Semiconductor Manufacturing Conference Proceedings* (October 1991).

J. McPhee, T. O'Toole, and M. Yedvabny, "Cooling the VAX 9000," *Electro/90 Conference Record* (May 1990).

J. McWha and P. Kouklamanis, "A Product Information Access System for Verification, Test, Diagnosis and Repair of Electronic Assemblies," *IEEE International Test Conference* (October 1991).

B. Mirman, "A Way to Avoid Stress Singularities in Multimaterial Elastic Bodies," *Transactions of Annual Meeting of the American Society of Mechanical Engineers* (December 1991).

T. Moore, "A Workstation Environment for Boundary Scan Interconnect Testing," *IEEE International Test Conference* (October 1991).

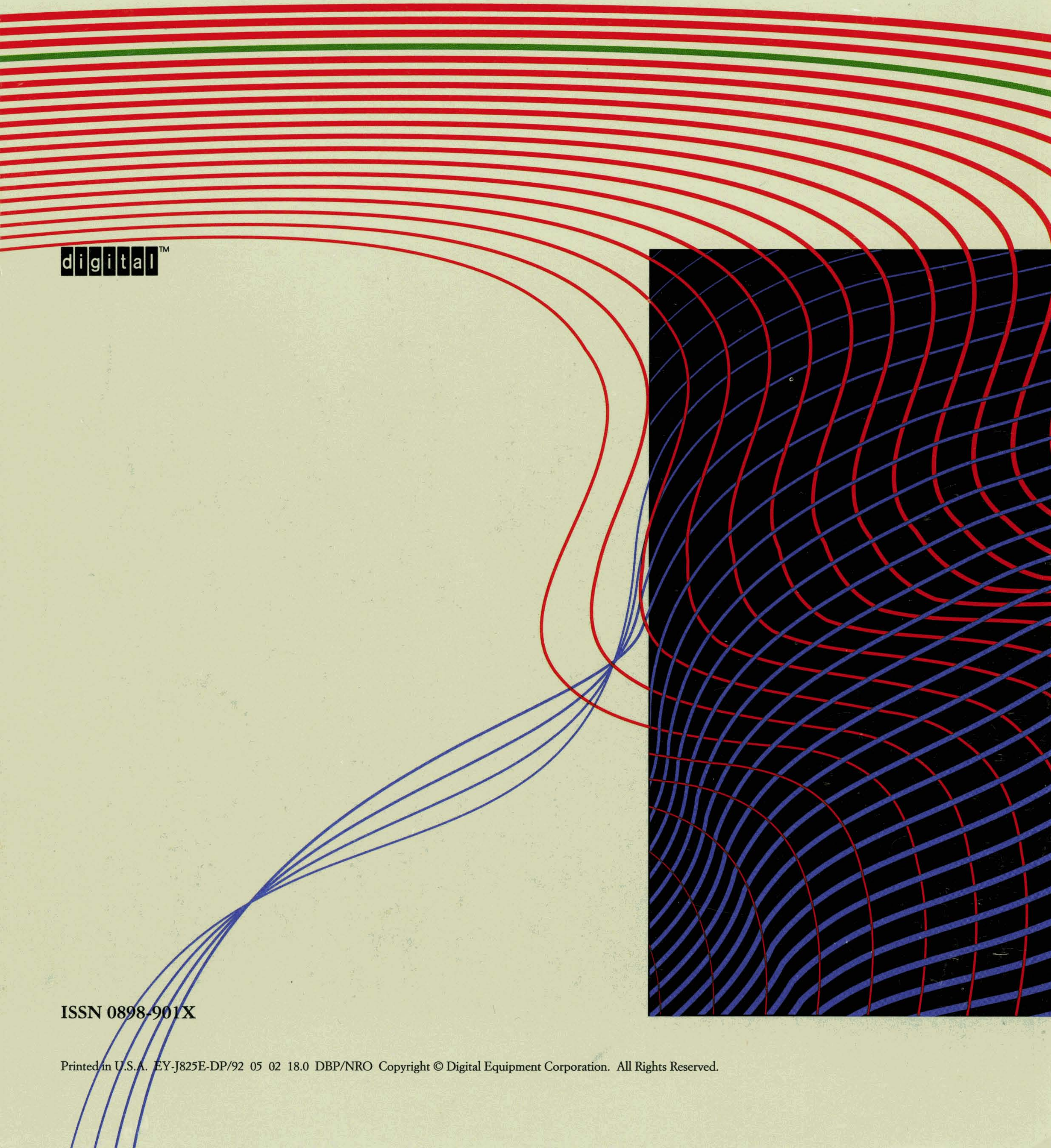
C. Pietras, "Cognitive Models of Planning in the Design of Project Management Systems," *Proceedings of the Human Factors Society Thirty-fifth Annual Meeting* (September 1991).

K. Ramakrishnan, "Dynamics of Congestion Control and Avoidance of Two-Way Traffic in an OSI Testbed," *ACM Computer Communications Review* (April 1991).

S. Rege, R. Kalkunte, R. Edgar, and A. Russo, "A High Performance FDDI Adapter for VAX Systems," *Thirty-seventh IEEE Computer Society International Conference* (February 1992).

M. Register, A. Rewari, and M. Swartwout, "The CANASTA Experience: Key Management and Technical Decisions in a Hybrid Expert System Project," *IEEE ACM International Conference on Developing and Managing Expert System Programs* (September-October 1991).

K. Symonds, M. Bahrami, and P. Skerry, "Functional Failure Analysis Using Photoemission Microscopy," *Proceedings of the Seventeenth International Symposium for Testing and Failure Analysis* (November 1991).



ISSN 0898-901X