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.

By Anthony J. Rizzolo, Elizabeth A. Brewer, and Martha A. Chandler

Introduction

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, RFC 1001 and RFC 1002 specifications.[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 server and management processes through message queues.

This process model is depicted in Figure 1 and is described in the following sections.

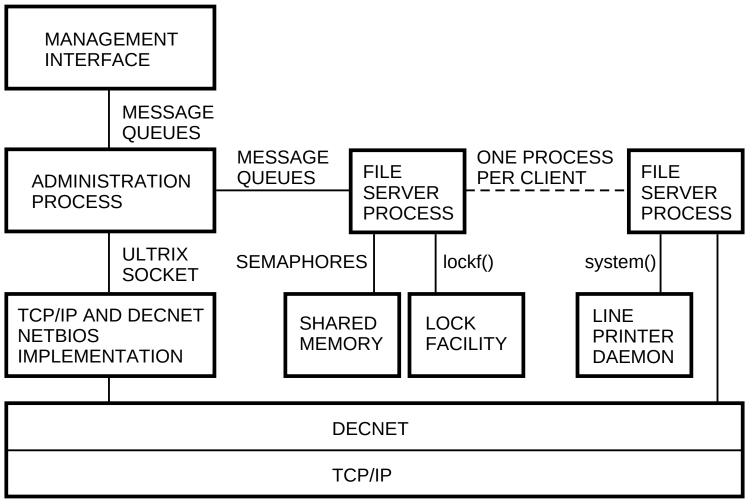

Figure 1 PATHWORKS for ULTRIX Process Model

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 previously stated, request processing is the main task of the administration process. 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 startup. 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 selected a PC-style format that uses pull-down 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.

```
/* 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

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 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 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.

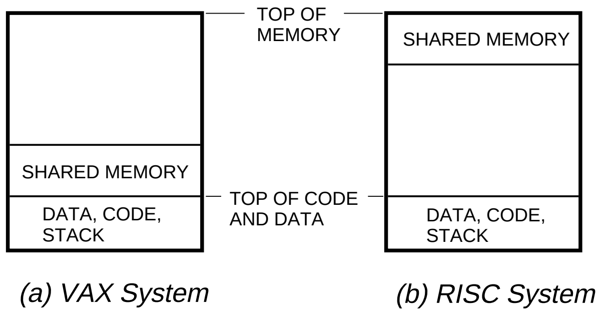

Figure 3 Image Memory Layout

To increase the data segment size in the VAX system, we replaced all malloc()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 Configuraion

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.
- 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.

Acknowledgements

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, Volume 2 (Maynard: Digital Equipment Corporation, Order No. AA-MF07A-TE, 1988).

Author Biographies

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.

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.

Trademarks