LBL-25770 c.2
UC-405

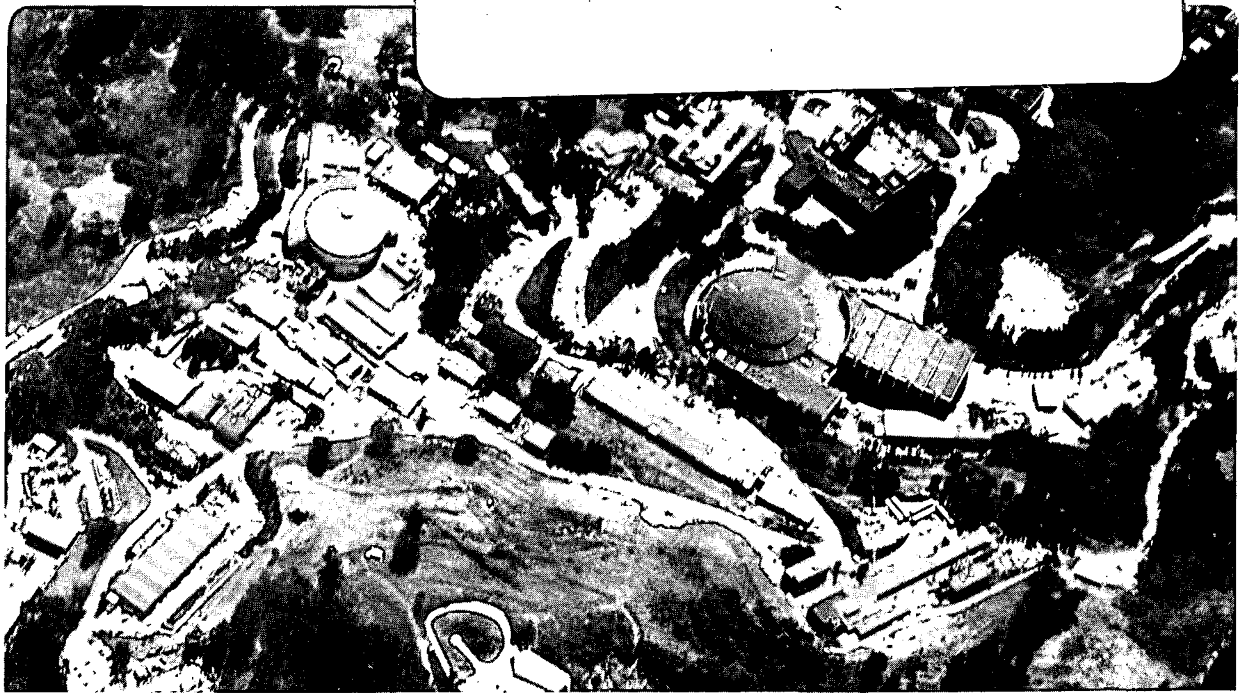
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Information and Computing Sciences Division

SOS — Stan's Own Server: an NFS File Server for the IBM PC

S.-M. Tan, H. Holmes, and C. Eades

August 1988



LBL-25770 c.2

Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

LBL-25770

SOS - Stan's Own Server
A NFS file server for the IBM PC

See-Mong Tan, Harvard Holmes, Craig Eades

Computer Science Research Department
Information & Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720

August 1988

This work was supported by the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

SOS - Stan's Own Server
A NFS file server for the IBM PC

See-Mong Tan, Harvard Holmes, Craig Eades

Computer Science Research Department
Information and Computing Sciences Division
Lawrence Berkeley Laboratory
Berkeley, CA 94720

ABSTRACT

SOS is a file server written to run on the IBM PC or compatibles. It conforms to Sun Microsystem's Network File System (NFS) protocol version 2. This paper discusses the SOS implementation and includes notes on portability, for programmers who wish to understand or modify it.

1. Introduction

In general, a network of workstations provides more computing power than a standalone mainframe does. Personal computers, however, face the problem of file storage and sharing, since some machines often do not have local disks attached (e.g. Sun 3/50s), while files needed by different users must be transferred from one machine to another, by some program such as **ftp**. This method of file sharing is at best still cumbersome.

Sun Microsystem's introduction of **NFS**, the Network File System, is designed to provide transparent access and sharing of files over a network. In essence, files exist on the locally attached storage device of a machine called the *file server*, whose job is to accept requests for operations on files from the other machines on the network, called the *clients*. The clients *mount* filesystems on the server. A filesystem is a directory on the server. The *mount point* on the client is the directory at which the client chooses to substitute for the server directory. Any operations on files below the mount point are trapped by the operating system and mapped into appropriate requests to the server. Sun Microsystem's has also recently introduced **PC-NFS**[1]. PC-NFS allows IBM PC's or compatibles with ethernet interfaces to become NFS clients on the network.

We have designed a file server conforming to the NFS protocol (version 2), that runs on the IBM PC. This allows a dedicated PC to perform a task otherwise required of a Sun workstation running NFS in kernel, or a VAX running a user-level NFS daemon. The original project intended for a PC to serve files from an optical disk to a heterogeneous network of Suns and PCs (with PC-NFS) as clients.

2. Managing Concurrency

The Sun NFS server consists of three different processes running concurrently. There is the **mount daemon**, **port mapper**, and the **server daemon** proper.

The mount daemon handles requests from a client for a filesystem directory to be mounted. If the request is valid, the mount daemon hands back to the client a *file handle*. The file handle contains context information, and the client must use it in further transactions with the server daemon. The handle is opaque and varies from server to server.

[1] PC-NFS is a trademark of Sun Microsystems, Inc.

The server daemon performs requested file operations such as reading and writing files, and reading of directories.

The port mapper always exists at a well known port, PMAPPORT, which is equal to 111. The server and mount daemons can theoretically be bound to any port, and the port mapper will answer queries from client machines regarding the port numbers of the other two daemons. In version 2 of the NFS protocol however, the server is expected to be bound to port 2049. Clients do not query the port mapper for the NFS port. This is a bug in the protocol which Sun Microsystem claims it will fix in the near future.

MS-DOS does not allow concurrent processes. **SOS** includes both port mapper and mount programs together with the server. Calls to these are by necessity sequential, not parallel. Since port mapper and mount requests are infrequent compared to NFS requests, this does not seriously hamper the server's performance.

3. File Handles and Inodes

Between a client and the server, a file is identified by a *file handle*. NFS protocol specifies the length of a file handle to be 32 bytes. In the Unix[2] implementation, the file handle contains (among other things) the inode number of the file.

MS-DOS does not explicitly support index nodes (inodes). Access to a file is almost completely **path driven**. Since the file handle must be only 32 bytes long, it is impractical to stuff the path name into the handle. A 32 byte long path would only allow the shallowest directory trees. Thus, **SOS** has an artificial inode interface to the files on disk. It assigns inode numbers to files as requests arrive for it to lookup or read. An image of the filesystem tree is built in memory, and a file's path name is reconstructed from its inode number by referring to the filesystem tree.

Since the inode numbers are artificial, they would not be preserved if the server crashes or is interrupted. Instead, the server writes to a file (*"inode.dmp"*) the path names of the files it had assigned inodes to. **SOS** will read and reconstruct the filesystem tree from both the export file and the inode dump file when it is next started up. This means that **SOS** is not really stateless, because it must preserve assigned inode numbers from one invocation to the next.

4. MS-DOS Constraints

MS-DOS was not designed for a multi-user system. Unix file attributes are inherently richer than can be put into an MS-DOS directory.

All files in an exported **SOS** filesystem are tagged owned by the superuser (user id of **0**). Further reflecting MS-DOS, files can exist in only three modes: a directory (global read, write and execute), a normal file, which is globally readable, writeable and executable, and a read only file, which is a normal file sans write permission.

This means that certain attributes cannot be set in exported files; for example, **chown**(1) will not work at all, and **chmod**(1) will only affect write access.

5. RPC/XDR Interface

RPC and XDR stand for Remote Procedure Call and External Data Representation respectively. RPC is a convention for calling remote procedures. A server program is identified by its **program number** and **version number**. Each procedure within a program is represented by a **procedure number**. For example, the read procedure in a NFS server is identified by this triple: 100003 (program number), 2 (version number), 6 (procedure number). A client wishing to read a particular file would send a request to the server in the form of the previous identifying triple, followed by the file's file handle. In order to decipher what is sent over the wire, both servers and clients use the XDR routines. XDR encodes and decodes data sent over the network in a

[2] Unix is a trademark of Bell Laboratories.

format not dependent on each machine's architecture.

Sun NFS runs on an RPC/XDR interface. We ported the public domain Sun server side RPC and XDR code to the PC. Only a few minor changes were made. One was allowing the procedure **svcudp_create()** to take two arguments, the second of which specifies whether to use a large (8800 byte) UDP send and receive buffer, or to use a small (400 byte) buffer. The port mapper and mount daemon use the small sized version, and the server daemon uses the large sized buffer. Also, the ported RPC procedures do not register services with the port mapper. The **SOS** port mapper knows of only two service ports, the mount daemon port and the server daemon port, which are compiled into **SOS**.

Only datagrams are used in **SOS**; the TCP interface was not moved. This mainly affects the port mapper. It cannot receive TCP requests.

6. IPC Interface

The project used a PC attached to the net with an Excelan EXOS card.

The Excelan EXOS package provided a socket interface quite similar to 4.2 BSD UNIX. A seperate module was written on top of the Excelan library (*"sock.c"*). This abstracts the socket interface which the rest of the program sees from Excelan specific details. Portability in the future to a different ethernet card will simply consist of rewriting this module in concert with the new library provided.

7. Some Difficulties Encountered

Before we moved development to a PC with an Excelan ethernet card, we tried to write the server on another machine running PC-NFS. PC-NFS provides a very congenial environment for writing a server. The toolkit provides XDR routines and client side RPC routines, as well as a good 4.2 BSD Unix socket and networking interface. Unfortunately, PC-NFS uses the local port number 2049 for its own transactions. In NFS protocol version 2, port 2049 is the de facto NFS port, and clients do not query the port mapper for the port number of the server. This means that any PC-NFS client cannot also be a server. We were forced to abandon this approach and use a PC with an Excelan card and Excelan software instead.

8. Server Performance

Some informal timings of server performance were done. We compared the speeds taken to access a large file by several different clients, from both a Sun 3/280 server and a PC running **SOS**. The reference **SOS** configuration was an IBM PC with a hard disk drive[3].

The Sun runs at 25 MHz with twenty times the bandwidth of the 4.77 MHz IBM PC. On the average, **SOS** on a PC was about nine times slower than the Sun, with a PC as a client. With a Sun client, **SOS** was some seventy times slower. The table below shows transfer rates in kilobytes per second.

Transfer rates in kilobytes per second

Clients	Servers		
	PC (SOS)	VAX (11/785)	Sun (3/280)
IBM PC	4.2	14.4	36.0
Compaq 386	4.4	17.7	125.9
Sun 3/60	2.9	100.7	220.0

Note that the performance of the Sun client with the PC server is not as good compared to the PC and 386 clients. This irregularity is explained by the fact that the Sun client tends to time

[3] We used a Microcode 20 Megabyte hard drive.

out before the PC server's reply, hence causing a double read request to be sent out.

Running the server in verbose mode (where the server advises of all incoming requests) slowed performance by about two-thirds. Running the server with files accessed from a virtual disk (RAM disk) created in memory gained a 20% increase in performance.

9. Improvements

The port mapper and mount daemons are incomplete. Clients currently cannot dump listings of ports or exported filesytems from the server.

The inode interface is inelegant. A better scheme would be to use the starting cluster numbers of files on disk to serve as inode numbers. We suspect that this would involve quite some work.

10. Acknowledgements

Fred Gey's generosity in loaning us his PC for SOS's development over one summer is greatly appreciated. Thanks to David Robertson for his bug free host to network and network to host data conversion procedures. We are also grateful to Bill Johnston for his help in troubleshooting PC-NFS's perculiarities.

11. Comments

Bugs and comments should be sent to:

stan@lbl-csam.arpa, stan@lbl-csam.lbl.doe.gov, lbl-csam.arpa!stan