REPRINTED FROM: **MINI-MICRO SYSTEMS**

SOFTWARE

Extending UNIX to local-area networks

PETER KAVALER and ALAN GREENSPAN,
Altos Computer Systems

The UNIX operating system—together with derivative products such as Microsoft Corp.'s XENIX—is rapidly becoming a standard for 16-bit microcomputers, providing a rich set of development tools and a solid base for running application software. A fundamental feature of both UNIX and XENIX is a hierarchical file system that hides hardware-dependent details, such as block size, record length, sector location and type of disk drives, from application programs and end users. This design provides a uniform user interface, but one that is confined to a single machine.

ALTOS-NET II software, which runs with XENIX on Altos Computer Systems' 8086-based microcomputers, extends the hierarchical file system to local-area networks. It allows transparent remote file access and remote processor execution, making it possible to build a network of XENIX-based systems that appears to be a single large computer. Network users can save substantial disk space by sharing files and can transport application programs to the network with minimal modification.

Enhancing the file system

The XENIX file system is an inverted tree with leaf nodes corresponding to data files, programs and devices; branch nodes corresponding to directories; and the root of the tree corresponding to a "root directory."

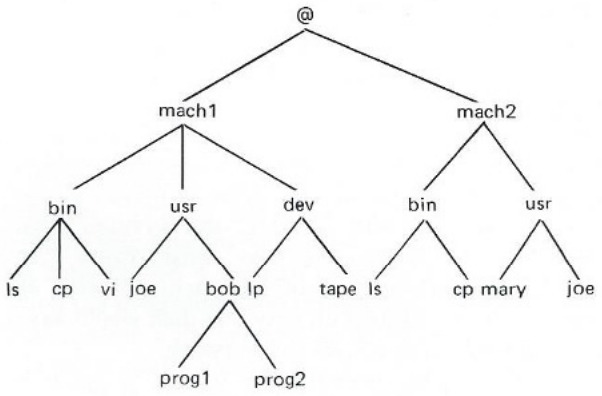

Fig. 1. The @ directory *is ALTOS-NET II's extension to XENIX's hierarchical file system. Traditionally, XENIX files on separate machines are linked by utilities that confine network functions to file transfer and remote log-on. But the @ directory establishes a new root on the tree, enabling all utilities and applications to function throughout the network.*

Users can access a node by specifying its location in the tree using a *pathname*—a sequence of node names separated by slashes. A pathname beginning with a "/" descends the hierarchy from the root directory. For example, the pathname */usr/bob/prog3* represents the file *prog3* within user Bob's directory. The command *cd* simplifies pathnames by designating a "current directory" as the starting point for pathnames not beginning with "/". If the current directory in the above example were */usr/bob*, the desired file could be represented by just *prog3*. However, in either case, file access is limited to a single machine. Inter-machine communication in UNIX or XENIX has required network utilities that have permitted file transfer and remote log-on, but little else.

ALTOS-NET II introduces a directory, called the "@ directory," one level above the root directory of each system in the network (Fig. 1). The @ directory enables users and application programs to access any file in the network using standard XENIX commands. For example, a user on the machine named *mach2* can use the pathname *@mach1/usr/bob/prog3* to access the *prog3* file on the machine named *mach1*. The user's only concern is the logical locations of the files within the file system hierarchy. Just as UNIX and XENIX hide the physical storage media and file-access methods from users, ALTOS-NET II hides the underlying network

ALTOS-NET II software enhances the XENIX kernel, allowing user-transparent access to remote files and processors

hardware and protocols. Moreover, while conventional networks require that application programs be rewritten to invoke network utilities, ALTOS-NET II runs application programs without modification or recompilation.

With ALTOS-NET II, there is little need to copy files from machine to machine; a single copy of a shared file need be kept on only one machine for all programs within the network to access it. By avoiding duplication of system utilities, disk space is freed for user programs and data. Since most microcomputer-based UNIX systems require about 8M bytes of mass storage for the full set of UNIX utilities, the savings from avoiding file duplication can be considerable. An ALTOS-NET II configuration of 10 machines with 80 percent of system utilities on only one disk would save as much as 60M bytes across the network.

ALTOS-NET II also permits the use of a low-cost disk-less workstation on the network. For example, the Altos 186-VDU has a 14-inch bit-mapped screen, an 80186 CPU and as much as 512K bytes of RAM—but no local disk storage. Such a workstation would be of little use on a conventional network because it could be used only to log onto another machine by emulating a terminal. But with ALTOS-NET II, another computer can act as a "file server" for one or more disk-less workstations by downloading the XENIX kernel and

System call	Purpose
File control	
access	Determine accessibility of file
chmod	Change mode of a file
chown	Change owner and group of a file
fstat	Set file status
ioctl	Control device
link	Link to a file
mknod	Make a directory or a special file
stat	Get file status
umask	Set file-creation mode mast
unlink	Remove directory entry
utime	Set file times
File access	
create	Create a file
open	Open for reading or writing
seek	Move read/write pointer
read	Read from file
write	Write to a file
close	Close a file
Process control	
chdir	Change default directory
chroot	Change root directory
exec	Execute a file
exit	Terminate process
fork	Spawn new process
setgid	Set group ID
setuid	Set user ID

The 24 system calls *that handle file control, file access and process control are intercepted by ALTOS-NET II, which determines if a call has network implications.*

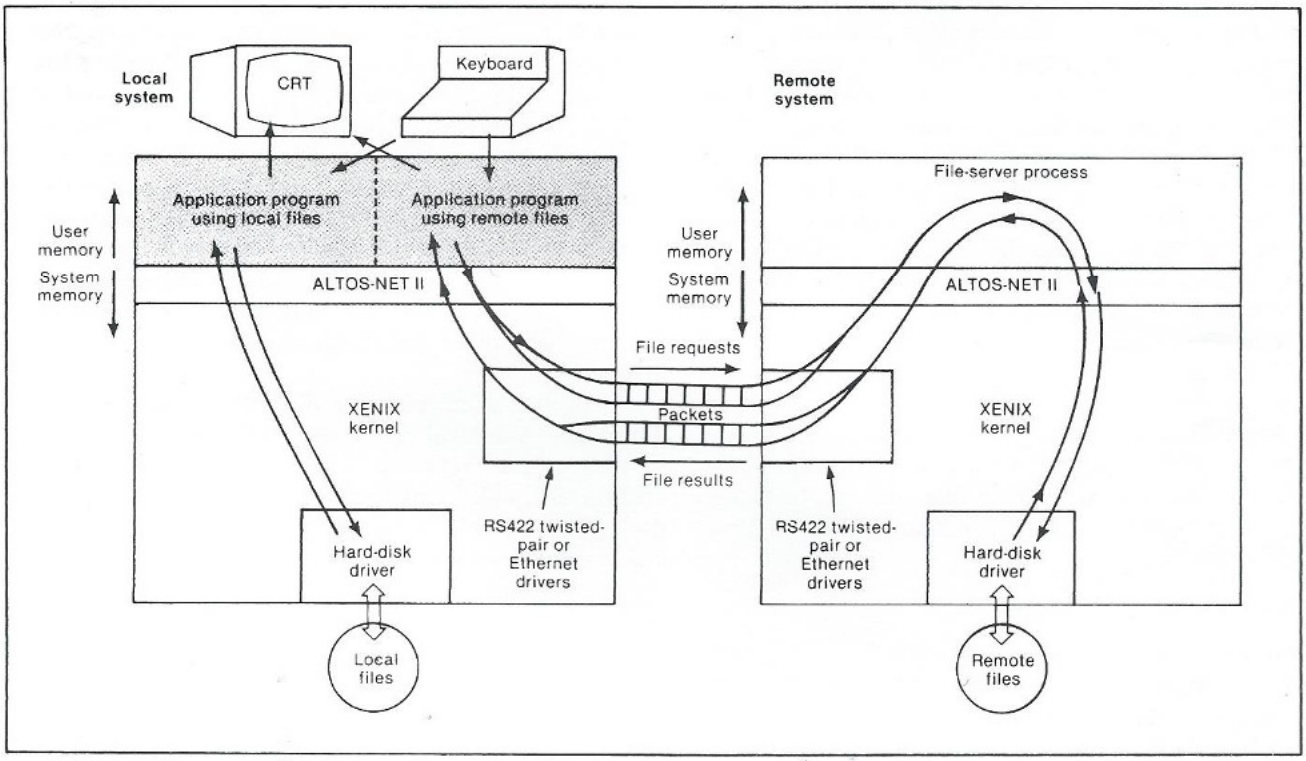

Fig. 2. Remote-file access *involves the same user interface as local-file access. ALTOS-NET II software intercepts each file request from an application program and determines whether the file is on the local or remote system. If local, the software passes the file request to the local operating-system kernel for normal processing; if remote, it transmits a packet to a file-server process on the remote system. From the standpoint of the remote system, the file-server process behaves as an application program requesting a local file.*

sharing its file system. A 40M-byte file-server machine attached to four disk-less workstations would more economically provide each with 10M-byte storage than would separate disk drives and controllers.

Design trade-offs

All programs running under UNIX must make calls to the operating-system kernel when they require main-memory allocation, process creation and termination or file access. A typical UNIX implementation supports about 60 such calls to the operating system, of which 24 have networking implications (see p. 198). ALTOS-NET II software residing in the kernel intercepts these calls immediately after the request is made and determines whether to process the calls locally or remotely (Fig. 2). If the request is for a local resource, it is passed to the kernel for normal processing. But if the user program is requesting a resource on a remote machine, ALTOS-NET II constructs a request packet and transmits it over the network. A file-server process on the remote machine receives the request, calls its own kernel for processing and passes the results back to the requesting machine.

For each process having at least one remote open file, a corresponding process, residing on the remote machine, services the requests on its counterpart's behalf. This process is created when a program opens its first file on a remote machine and is terminated when the program closes its last remote file. Moreover, the creation and termination of processes occur automatically; neither users nor application programs are aware of it. There is a brief delay when a new file server is created; after that, the lag time resulting from network transactions is negligible (see "Benchmarking ALTOS-NET II," left).

Altos chose the implementation technique of invoking file-server processes that interact with a layer of software inside the OS kernel.

A special concern arises in implementing record and file locking in ALTOS-NET II. If a remote system goes down before releasing a locked resource, each file-server process periodically polls its corresponding requester process. If the requester fails to answer several consecutive polls, system failure or cable disconnection is assumed, and the server process releases all locks and terminates.

Altos chose the implementation technique of invoking file-server processes that interact with a layer of software inside the operating-system kernel over several other alternatives. One alternative could link the kernels on the various machines and forgo using file-server processes. Under this scheme, UNIX i-nodes, the primitive and normally hidden data about the location of files on a disk, would be passed from kernel to kernel. This approach suffers from the heavy amount of i-node traffic on the network, which could cause lengthy delays. Also, by passing UNIX internals over the network, the inclusion of other operating systems on the same network is all but impossible.

Another possible approach uses file-server processes but moves the interface layer from the kernel to a library of routines that is linked with each application program. Although such an implementation appears attractive because of the relative ease of debugging

BENCHMARKING ALTOS-NET II

Several performance measurements have been made on an operational ALTOS-NET II system. The system contained two Altos 586 systems connected by an RS422 twisted-pair cable. The Altos 586 system is based on a 10-MHz 8086 processor; the standard system comes with 512K bytes of main memory, an 8089-based hard disk controller, a 10M-byte hard disk (with an average seek time of 85 msec.) and a Z80-based communications processor. The Z80 controls the transmission and reception of packets on the network cable, and as many as five ASCII terminals attached to the system. The network uses the SDLC mode of the Zilog SIO chip and standard RS422 drivers to transmit data at 800K bps.

The measurements show the time required to load and run the *cp* file-copying utility. Benchmark 1 is the "control case" that employs no network requests. Benchmarks 2, 3 and 4 are "worst-case" network tests in that virtually no local processing is done between disk and network requests.

Benchmark 1:
cp big1 big2 — Copy 315K bytes from big1 to big2 on the same machine.
CPU time = 6.9 seconds; Real time = 32 seconds.

Benchmark 2:
cp big1 @dept2/big2 — Copy 315K bytes from the local machine to a remote machine.
CPU time = 10.7 seconds; Real time = 48 seconds.

Benchmark 3:
cp @dept2/big2 big1 — Copy 315K bytes from a remote machine to the local machine.
CPU time = 6.6 seconds; Real time = 43 seconds.

Benchmark 4:
cp @dept2/big2 dept2/big2 — Copy 315K bytes from a remote machine to the same remote machine.
CPU time = 10.2 seconds; Real time = 80 seconds.

Benchmarks 2 and 3 indicate that it is slightly more efficient to read from a remote system than to write to a remote system; this is due to the effects of XENIX's read-ahead buffering algorithms. Benchmark 4 shows the worst-case performance that can be expected from a disk-less workstation.

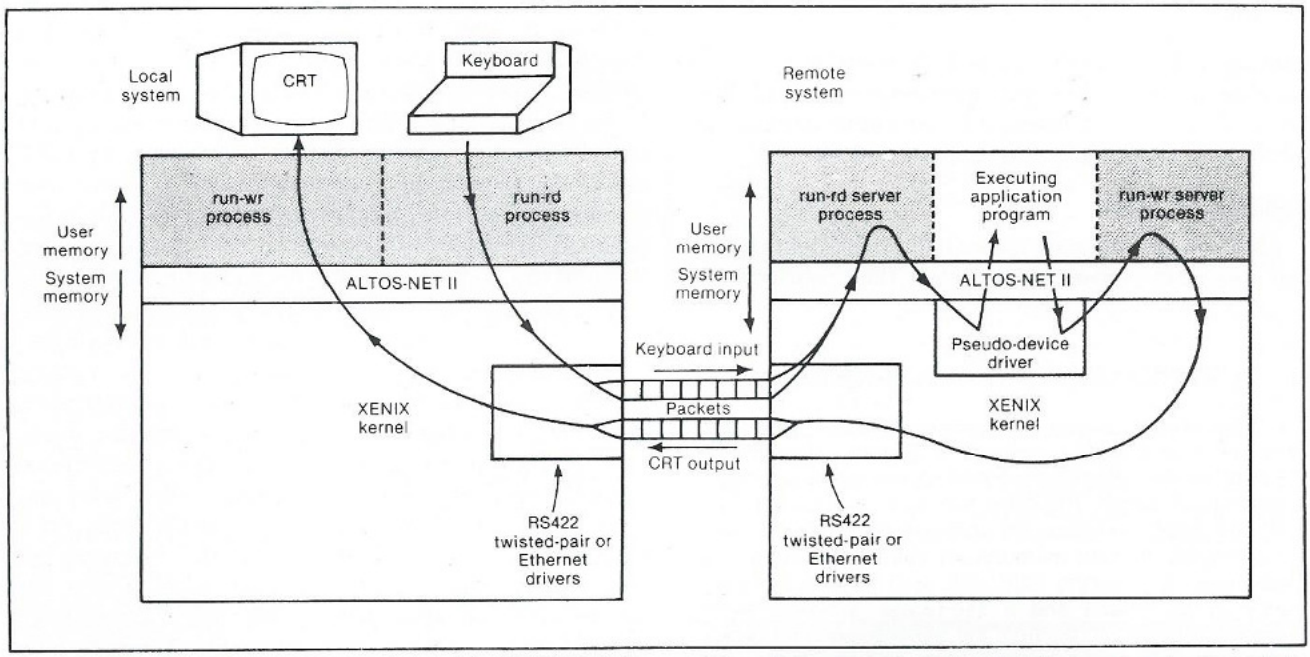

Fig. 3. Remote-program execution *is directed by ALTOS-NET II's* run *utility, which creates four processes that read and write packets transmitted between systems. Once the* run *utility is invoked, the user interface for remote processing is identical to the user interface for local processing. In the remote system, a pseudo-device driver communicates with the application program, functioning as a "logical" keyboard and CRT for the application.*

software outside the kernel, the interface layer of about 8K bytes of code would have to be included with each application, causing a significant reduction in memory available for user programs. Moreover, all applications would have to be relinked with the new library before they could function in the network.

Remote-processor execution

ALTOS-NET II supports remote-processor execution, which works with remote-file access. A network can contain as many as 32 Altos 8086- and MC68000-based systems. Mixing processors allows specialization; for example, some processors can be configured with large memories, while others offer floating-point capability.

An ALTOS-NET II utility called *run* executes a program on a remote processor yet makes it appear as if it were executing locally. The utility is especially applicable to large networks containing a variety of computers that are each capable of running a different set of applications. While running a program on a remote computer, a user can access files from yet a third machine.

The *run* utility works by initiating four processes and using a pseudo-device driver to exchange information between local I/O devices and the remote processor. When a *run* request is made, the utility sends a start-up packet to the server machine, and both the *run* utility and *run* server processes split into two subprocesses: *run-wr* and *run-rd* on the requesting machine and *run-wr-server* and *run-rd-server* on the server (Fig. 3). The *run* utility then sends a set of packets to the server containing the name of the program to be executed, the user's security information and the user's working directory. All keyboard input from the requester is constructed into a packet and sent to the *run-wr-server* on the remote machine, which writes it to a pseudo-device driver in the kernel. Similarly, screen output is sent from the driver to the *run-rd-server*, which constructs a packet and transmits it back across the machines to the *run-rd* process. In both cases, the pseudo-device driver acts as the stand-in for the user's terminal.

At the lowest level of communication, both transparent file access and remote-processor execution use a subset of Xerox Corp.'s Internet packet-exchange protocol to assure reliable data transmission. Because this protocol is relatively simple, ALTOS-NET II implements it entirely within the kernel for maximum network efficiency. □

Peter Kavaler is product manager for networking communications, and **Alan Greenspan** is a software engineer with Altos Computer Systems, San Jose, Calif.