CPNETDC.WS4

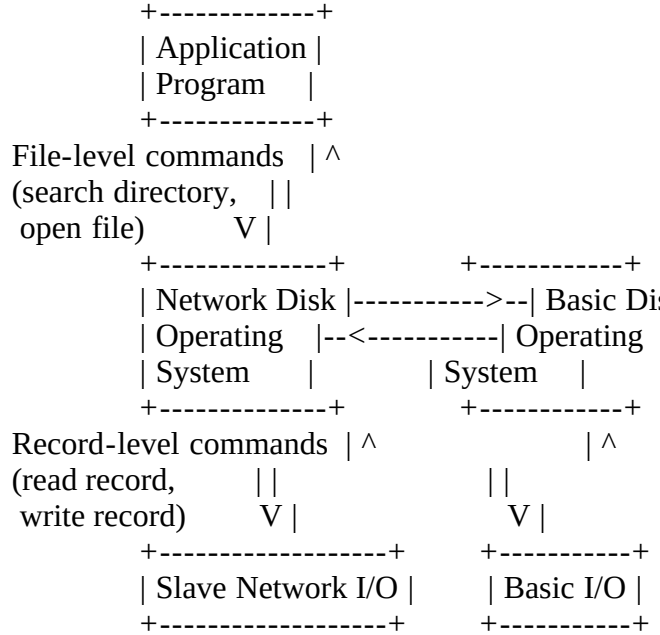
- "Network software borrows design from microcomputer operating system"
  Thomas Rolander, Randall Baird, & John Wharton
  "Data Communications", Vol.11, No.13, December 1982, p.123

(Retyped by Emmanuel ROCHE.)

Computers produced by different vendors are seldom fully compatible. Obviously, a 5.25-inch diskette drive will not accept an 8-inch diskette intact, but often the conflict is far more subtle (see "Factors affecting diskette compatibility" at the end of this article). Microcomputer networking offers a clean solution to such incompatibility problems: Directly transmit data, rather than physically transport magnetic or paper copies. Much progress has been made in the field of networking hardware. However, far less effort has been directed toward developing useful software interfaces to support this hardware.

Ideally, an application program should be able to open, read, write, and close a file on a local or remote drive with equal ease. It takes a significant amount of software, though, to transform high-level function calls into relatively low-level network commands. In the past, such software was reinvented by each system designer. Digital Research has addressed this problem with a general-purpose network operating system called CP/NET, which operates in conjunction with the widely-used CP/M control program (see "What is CP/M?" at the end of this article).

CP/NET provides an intermediate level of software that transforms high-level service requests from application programs into calls to low-level network I/O driver routines (see Figure 1). The application program interface defined for CP/M has been retained by the network software, so unmodified programs originally developed for the former operating system can now operate in network environments.



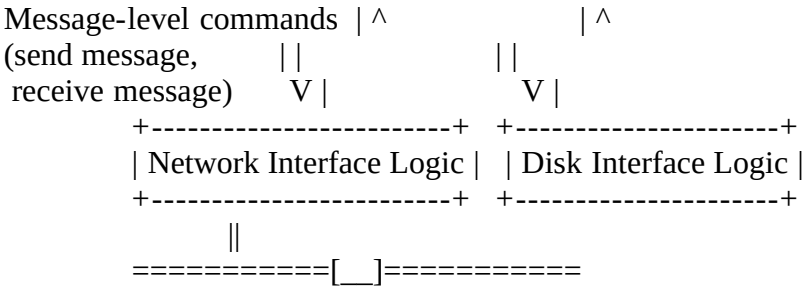


Figure 1. Interface software. CP/NET translates high-level operating system calls like "search directory" or "open file" into low-level network operation like "send message".

The logical portions of the network interface are isolated from the physical driver routines themselves, allowing designers to adapt the network software to a variety of network technologies by rewriting just the physical network driver routines. The network can thus be adjusted to handle different data rates, protocols, and network topologies.

Some network configurations allowed by today's technology are shown in Figure 2. In each case, one or more nodes act as hosts for shared network peripherals and are called network servers. The other nodes are called network requesters, and correspond to individual workstations with their own local disk drives.

(Not shown: Star, Loop, and Multidrop topologies.)

Figure 2. Architectural flexibility. A CP/NET network consist of at least one network server node and many network requesters. Software accommodates various network technologies, including Star, Loop, or Multidrop topologies.

An operator at a workstation has access to a number of logical peripheral devices, including a console (CON:), line printer (LST:), and up to 16 disk drives (A: through P:). These correspond to standard physical peripherals of a conventional standalone CP/M microcomputer. Each logical device may be associated with either a peripheral attached directly to the node, or with a shared peripheral accessed via the network.

Consider the network shown in Figure 2a (Star topology). The microcomputer at node 02 (network requester) is able to reference its own console and diskette drives A and B. However, the operator can reconfigure the network to take advantage of any of the peripherals connected to node 01 (network server).

A utility program called NETWORK changes logical devices to physical device mappings. For example, the commands:

```
A>network h:=c: (01)
A>network lst:=01 (01)
```

will reconfigure the network so that logical drive H: and the line printer device are assigned to the network; drive H is mapped onto hard disk drive C, and the line printer is mapped onto printer 0. Users or software routines at node 01 may then access drives A, B, and H, and the printer as if all were physically attached to the CPU at that node. Figure 3 shows how the CPNETSTS

utility reports the network status after these and other device reassignments.

```
A>cpnetsts

CP/NET 1.2 Status
=================
Requester ID = 02H
Network Status Byte = 14H
Disk device status:
 Drive A: = LOCAL
 Drive B: = LOCAL
 Drive C: = Drive A: on Network Server ID = 01H
 Drive D: = Drive B: on Network Server ID = 01H
 Drive E: = LOCAL
 Drive F: = LOCAL
 Drive G: = LOCAL
 Drive H: = Drive C: on Network Server ID = 01H
 Drive I: = LOCAL
 Drive J: = LOCAL
 Drive K: = LOCAL
 Drive L: = LOCAL
 Drive M: = LOCAL
 Drive N: = LOCAL
 Drive O: = LOCAL
 Drive P: = LOCAL
Console Device = LOCAL
List Device = List #0 on Network Server ID = 01H
```

Figure 3. Typical network configuration. The CPNETSTS utility reports the current configuration of logical devices, such as disk drives and printers available to the network node.

When a program accesses a network resource, the console command to edit a text file on disk might be:

```
A>ed h:text.fil
```

(as with standalone CP/M). The first two characters ("A>") are a command prompt indicating that drive A will serve as the default drive for locating the desired file. The file prefix "h:" indicates the logical drive on which the text file should be opened.

The Console Command Processor program first loads the editor program from local drive A. The editor then calls an operating system procedure to open the specified file. This is the same as with standard CP/M. The network does not simply access the specified drive; rather, it first searches an internal device-configuration table to find the current mapping of logical device H.

Figure 4 shows that H is assigned to a drive attached to the network server, so that TEXT.FIL must be accessed via the network. A message is transmitted to the server specifying the type of operation requested (Open File), the physical device involved (server drive C), and the target file name (TEXT.FIL).

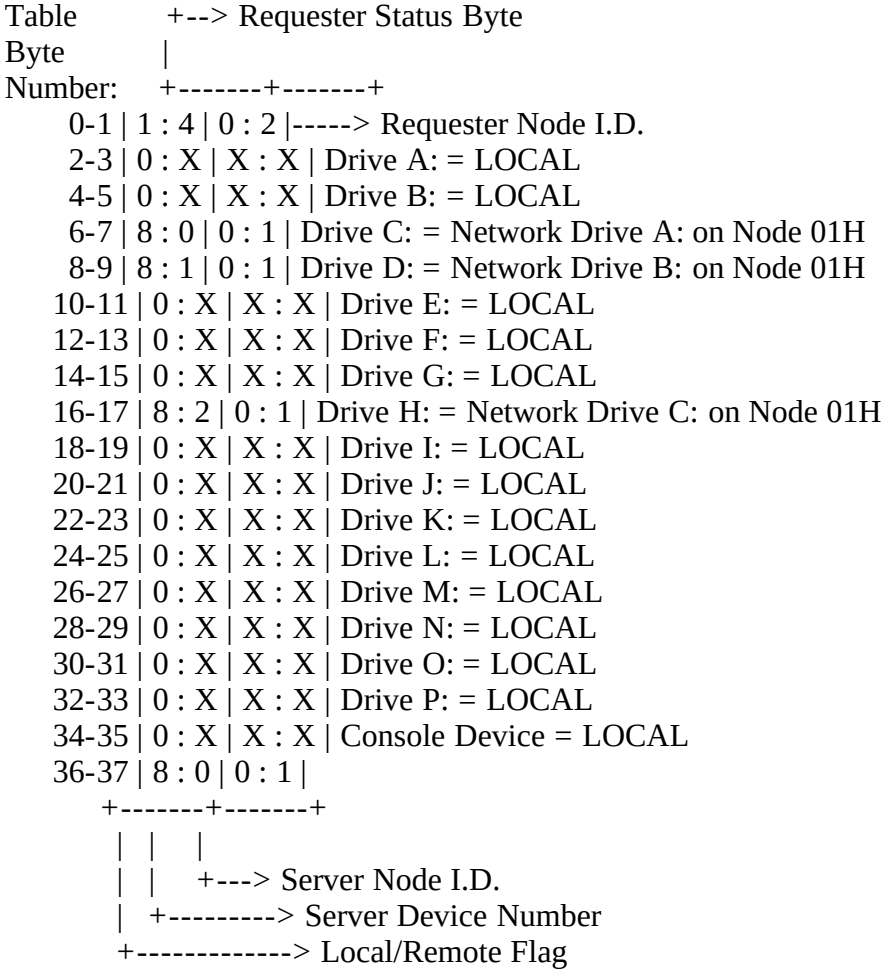
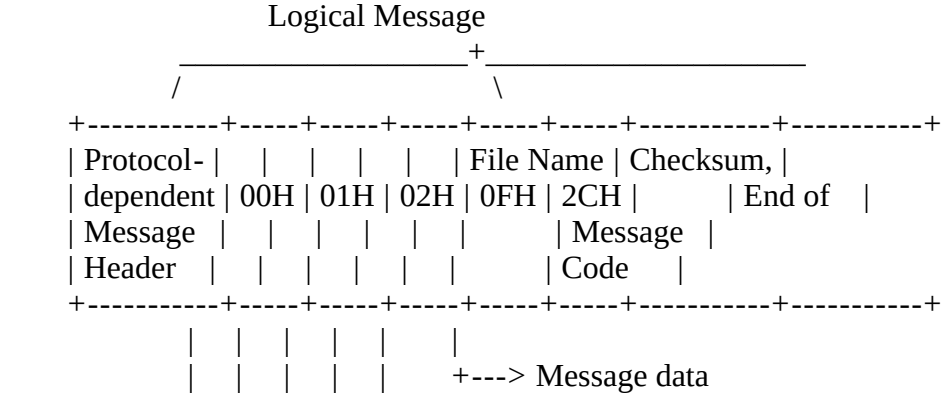


Figure 4. Table-driven. Before servicing operating system calls, CP/NET examines an internal device-configuration table to determine where each logical device is located.

The server receives the message, finds the appropriate file, and responds by transmitting a completion code to the requester. The editor may then read or write data from the file by calling other operating system procedures. Each procedure produces a similar sequence of network transactions.

The messages constructed and interpreted by the network software follow the format shown in Figure 5. The length of the data field varies depending on the type of operation. These are called "logical" messages, since they include just enough information to complete the network transaction, without framing characters or error detection.



```
|   |   |   |   +---> Data field length-1
|   |   |   +---> CP/NET function code
|   |   +---> Source node I.D.
|   +---> Destination node I.D.
+---> Request format code
```

Figure 5. Standard message format. Network nodes communicate via messages of standard format. Physical I/O driver software translates CP/NET logical messages into physical messages suitable for transmission by adding fields, such as those for physical and data link protocols, and for error checking.

As Figure 5 suggests, the network-device-driver software for a particular computer will generally construct a longer, "physical" message by combining the logical message with some sort of header and checksum, according to the physical and data link levels of transmission protocol. In many cases, the entire logical message becomes just the data field of the physical message. The standard logical message format contains enough information about source and destination nodes, and message length, to derive a data packet for any of the popular protocols, including HDLC, X.25, Ethernet, and Omninet.

Networking benefits

The above example shows the general case of an application program -- such as an editor -- accessing files on a remote network node. In the same way, a simple file-copy utility program could transfer data between local and remote drives, eliminating the need to transport diskettes. With a remote line printer appearing as an LST: device, hard-copy listings could be generated remotely without first putting the file on a remote drive.

Networking reduces costs. Adding network interface hardware and software costs far less than duplicating printers and large hard disks. A minimum CP/NET configuration would consist of the microcomputer, a console, and one diskette drive from which initially to load the operating system software.

In fact, networking can even eliminate that one local drive, further simplifying a minimal workstation. An operating system such as Digital Research's CP/NOS, derived fron CP/NET, puts the entire network operating system into a ROM or EPROM, instead of on a diskette. In this case, all application and utility programs will be retrieved from network servers.

This makes a complete workstation nearly as inexpensive and simple to manufacture as current microprocessor-based CRTs. In contrast to conventional time-sharing networks, though, the CPU in each workstation would perform its own processing. One version of CP/NOS reduces memory requirements by putting into ROM only the software that initially logs onto the network. After establishing communications, the rest of the operating system is downline-loaded through the network itself.

In addition to password protection schemes to increase the security of shared files, CP/NET also enforces certain access rights once a shared file has been opened. A file may be opened in either "locked" or "unlocked" mode. When a

locked file has been opened, only one program at a time may access that file. Unlocked files may be opened and accessed from several network nodes simultaneously. In addition, files may be defined as read-only or read/write.

For further security, the network allows common access only to files on network server nodes. The standard logical network protocol does not allow any other node to initiate a transaction involving a network requester. Consequently, files on requester-node drives cannot be accessed remotely without intervention by an operator.

Moreover, the requester-node network software is not time-shared, so it is very difficult to inject a spurious instruction stream into the CPU. Thus, an operator at a requester workstation is assured that all files on his diskettes are free from electronic tampering via the network. File security then becomes a concern for which there are more conventional solutions: Sensitive diskettes must be removed or locked up when not in use.

There will, of course, be times when one legitimately needs to review a file kept at a co-worker's desk. If so, the operating system's message facilities could be used to mail an electronic memo asking for a copy of the file. At that point, it would be up to the operator to determine whether the request is justified before taking the explicit steps to make a temporary copy of the file accessible via one of the shared drives.

Special requirements for shared files

Multi-user files normally allow concurrent file sharing only by not allowig the file to be modified. If two users attempt to update the same file at the same time, disk writes could collide at the same sector, leaving the file in an indeterminate state. Worse still, an operating system cannot always tell when a conflict has occurred.

Disallowing write permission would be particularly unfortunate where microcomputer networking was part of a multi-user database management system. To facilitate multi-user access to large files, the network has security mechanisms at the record level to ensure data integrity in shared, unlocked files.

Before altering a shared file, an application program must call the operating system lock-record function, which returns a go/no-go completion code. If the record in question has already been locked, the application program must wait until whoever is accessing the record has finished. If the record is available, the operating system locks it up and returns a successful completion code. After updating the file, the application program calls the unlock-record routine to release the record to any other programs waiting for it.

Of course, these features would be irrelevant if application programs could not conveniently take advantage of them. File sharing and record locking are currently supported by compilers for two high-level application languages, PL/I-80 and CB-80. The former implements the full general-purpose subset G of PL/I; the latter translates programs written in a commercially-oriented

superset of standard BASIC called CBASIC. Both generate 8080 machine code for run-time efficiency.

Network implementation

A number of microcomputers already on the market will support CP/NET. Even if a manufacturer does not offer a networking option, however, it is possible for users to design their own network interfaces. Though custom interface development has certain disadvantages over standard vendor-supported products, it might still be the most attractive alternative. For instance, this might be the only way previously purchased microcomputers can be retrofitted or tied to an existing mini-computer network.

Adapting a microcomputer to network software involves both hardware and software modifications. Additional hardware must be installed so the computer can connect to the network cable. If the microcomputer uses one of the standard backplane buses and a widely-used network protocol has been selected, suitable off-the-shelf hardware might be available. Intel offers a number of add-on printed-circuit boards that interface its Multibus (IEEE-P796) to synchronous networks and Ethernet, for example. Other vendors have similar products for the S-100 Bus (IEEE-P696) standard.

If the bus or network interface is non-standard, custom hardware may be needed. Fortunately, integrated-circuit manufacturers have developed a variety of sophisticated and versatile large-scale integration devices that give the hardware designer a good start.

In lieu of plug-in interface cards, protocol converters could be inserted between the microcomputer and the network. These converters could use the RS-232-C interface resident on most microcomputers.

One such converter is the ULCnet adapter manufactured by Orange CompuCo. Each self-contained box, about the size of a thick book, contains its own power supply to keep the network "alive" when individual workstations are turned off. Simular units are tied together in a multidrop configuration with standard modular-jack telephone cable and fittings. Messages between units follow a proprietary contention-type protocol, rather than any of the industry standards, but this may not matter within most office environments.

Once the necessary interconnect hardware has been designed or acquired, network interface software must be developed. This software consists of the seven separate subroutines listed in the table, collectively dubbed the Slave Network I/O System (SNIOS).

Five of the routines are quite straightforward: Network Init and Network Error ready the interface hardware for operation following error conditions or startup, respectively; Network Status and Config Table Addr let the network operating system access the data structure maintained by the SNIOS; the Network Warm Boot routine, which is called following the completion of each application program, should perform any periodic "housekeeping" functions required by the hardware. For instance, the warm boot routine could poll the network to check for mail posted to that device. If no housekeeping is

desired, this routine does nothing.

More complex processing is required by the Send Message and Receive Message routines. The former must accept a string of data bytes (the logical message discussed above) from the operating system, transform it into a physical message appropriate for the network hardware, and transmit the message to the indicated destination node. The latter routine performs the inverse operation.

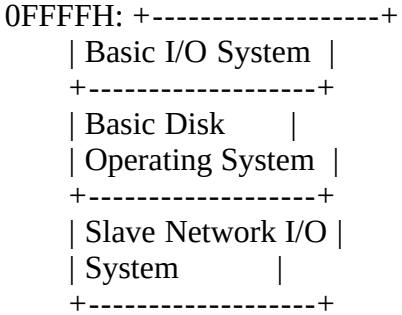
Requesters always perform these functions in pairs: A service request message is first sent onto the network, then the requester software waits until the response has been received. If the network protocol requires that error checking and/or correction or retransmission be performed by software (that is to say: if these operations are not handled automatically by the interface logic), Send and Receive should also incorporate these functions.

The standard CP/NET distribution diskette contains SNIOS routines for a simple point-to-point asynchronous protocol through an existing RS-232-C port. Source code for the routines is provided as an example and starting point for designers wishing to adapt the network to any particular interconnect hardware and protocol.

The major portion of the CP/NET requester software is called the Network Disk Operating System (NDOS). This module does not need to be modified, so it is provided only in object code form. It is sometimes called the logical portion of the network operating system, since it performs no direct intput or output. The NDOS calls the various SNIOS modules for all physical network I/O, and will therefore be directly compatible with any hardware for which a SNIOS has been properly developed. (When an application program accesses disk drives or peripherals that appear to the node as local, the NDOS calls appropriate routines within CP/M.)

Custom SNIOS routines may be developed on any CP/M-based computer, using the 8080 assembly language and a relocating macro-assembler. The task is simplified considerably by studying the code examples provided. The code may be initially debugged and tested in conjunction with the network hardware using either a software debugging program or a hardware debugging tool, such as Intel's In Circuit Emulator.

After initial testing, the SNIOS and NDOS modules and the NETWORK utility programs are copied onto a CP/M diskette for the target device. The CP/M command program CPNETLDR then loads the two object modules into memory just below CP/M itself, initializes the network interface, and invokes the network console command processor. At this point, device memory will be allocated (see Figure 6).



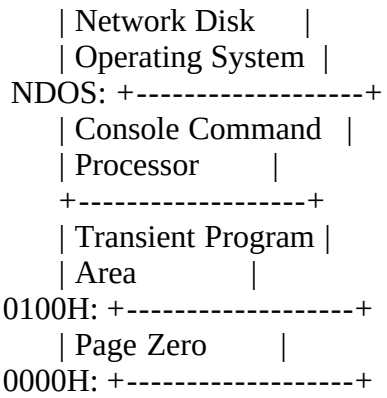


Figure 6. Memory Layout. CP/NET operates in conjunction with CP/M. Operating system calls not involving the network are passed onto the Basic Disk Operating System (BDOS).

Also available is standard CP/NET Server software, which runs under the MP/M-II multiprogramming operating system. Message formats and functions are documented to permit users to develop their own network servers under other mini-computer or mainframe multi-tasking operating systems.

Appendix

Factors affecting diskette compatibility

The wide variety of diskettes and the resulting incompatibility between microcomputers has created the interest in networks such as CP/NET. This incompatibility shows up in differences in diskette drive design, physical diskette formatting, and logical file structure.

There may be either a single "index" hole next to the main "spindle" hole to let drive electronics sense each revolution of a "soft-sectored" diskette, or a series of holes which mark the start of each "hard" sector. One or both sides of the diskette may be available for data storage. Some 5.25-inch diskette drives even differ in the number of tracks and track spacing they support.

Differences in physical formatting relate to the controller eletronics, rather than drive or media differences. Otherwise-identical diskettes may differ in how the molecules of their magnetic emulsions have been aligned to define bits and sectors. Recording density -- single or double -- is determined by the number of microseconds between interspersed clock pulses (synchronization) and data bits. Double-density recording even employs two different (and incompatible) algorithms to determine when to omit clock bits: modified frequency modulation (MFM) and modified-modified frequency modulation (M2FM).

On the matter of sector length and, consequently, how many sectors fit on each track, most large-scale-integration diskette controllers let the designer decide how many bytes constitute each diskette sector. Commonly selected values are 128, 256, 512, 1024, 2048, 4096, and 8192. A double-density 8-inch diskette would then contain 52, 26, 15, 8, 4, 2, or 1 sector(s) per track,

respectively.

Each sector size/count combination leaves a certain fraction of the track's potential storage capacity unused. This space is spread more or les equally among various gaps between sectors and the ID and data fields within each sector. Gap sizes are not critically important, but differences can cause erratic operation of poorly designed or maladjusted data-recovery circuits.

There is some uncertainty about what a diskette controller should do just after a sector of data and its error-checking code have been written. Most controllers record an immediate sequence of 1 bit (postamble); some do not. When a sector is re-read, some controllers require the postamble; most do not. Problems arise when a controller that requires the postamble tries to read a disk recorded without it. Such reads usually fail, but not always.

Suppose, though, that different manufacturers agree to a rigorous set of specifications, such that their diskettes have the same physical format. (It helps, of course, if one of the manufacturers is IBM.) Computer A could then read diskettes written by computer B.

This does not mean computer A could necessarily make sense of what it read, due to diskette differences of a third type. The way in which an operating system allocates sectors between dedicated functions, file directories, and file data, and the definition of diskette data structures needed to retrieve file data, is called the operating system's "logical file system". Independently developed operating systems have incompatible file systems, so computer A would not know where to look to retrieve computer B's files. (One could write a program to read "foreign" diskettes, though, if one understood their file system's data structures.)

With so many possible areas for conflict, it is remarkable that different manufacturers could ever achieve diskette interchangeability. Yet this has happened. When CP/M was first developed, the only commonly available diskette hardware handled 8-inch, single-density, single-sided, 128-byte soft-sectored diskettes. CP/M's file system was therefore originally designed around this physical format.

As diskette technology improved, CP/M was adapted to take advantage of the greater capacity of newer drives. Even so, many hardware designers decided that more sophisticated diskette controllers should also be able to handle the original single-density format.

Most microcomputers with 8-inch drives can thus switch into a special hybrid mode to read standard CP/M format, which has become an accepted medium for interchanging program and data files. (Ironically, IBM originally developed the flexible diskette for precisely this pourpose -- to replace punched cards as a medium for transporting data between computer peripherals and computers.) This, in turn, allowed independent software vendors to sell a single "flavor" of diskette that would satisfy most of the market.

No similar standard has evolved for 5.25-inch diskettes, much to the chagrin of software developers, vendors, and buyers. To satisfy a significant fraction of the 5.25-inch market, a software vendor must manufacture diskettes in more than a dozen formats.

What is CP/M?

CP/M, a "control program for microcomputers", was one of the first general-purpose operating system for 8-bit microprocessors. It consists of a disk file system that resides in main memory, a transient console-command interpreter, and a variety of utility programs loaded from disk that edit, assemble, and debug application programs. The objective of CP/M is to provide a machine-independent environment for the execution of application programs, allowing programs to be transported between different machines in object- or source-code form, without modification.

To maximize the memory available to application programs, the functions supported by the operating system are quite straightforward. Console input may be read a line at a time, with various control characters recognized to allow simple line editing. Disk files must be accessed in fixed 128-byte records. However, the functions preserve the flexibility to get at low-level I/O operations while retaining machine independence.

Other operating systems derived from CP/M support more sophisticated user needs. CP/M-86 keeps all the functions of the 8-bit version, while adding new capabilities appropriate to the architecture of the 16-bit 8086 and 8088 microprocessors. Application programs load dynamically into available memory, and new operating system routines support segmentation and manage the allocation of up to 1 Megabyte of main memory. Concurrent CP/M allows the user to execute more than one program simultaneously.

CP/M and CP/NET maintain strict separation of logical and physical portions of the software. The abstract operations performed by the operating system -- interpreting operating system calls, maintaining file system data structures, generating network messages -- are combined to form one module; all I/O functions form another. The logical modules never perform any direct input or output but, instead, call routines within the physical module.

The separation of functions was originally intended to facilitate the installation of standard software on widely varying hardware. The bulk of the software was the same for all users, with only the hardware-specific I/O routines adapted to new computer designs.

The separation of functions also opened the door to operating systems such as CP/NET, which provide the same logical functions as CP/M to the application program, but with a radically different physical implementation. An application program originally intended for use in a simple standalone CP/M-equipped computer can now, without modification, access devices scattered throughout a local network, just by replacing the operating system on which it runs.

EOF