Workstations and mainframe computers working together

by J. K. Kravitz
D. Lieber
F. H. Robbins
J. M. Palermo

The history and design philosophy of a research project that produced a prototype software package are described. The package provides cooperation between large-scale mainframe computing systems and personal desktop workstations. Also discussed are the features of PC/VM Bond, the product that grew out of the research project.

In recent years, the number of intelligent workstations in use throughout the world has grown at an ever-increasing rate. This fact is also true at the Thomas J. Watson Research Center, a major research facility of IBM. For many years, the Watson Research Center has been at the leading edge in the use of interactive computer systems (containing a very large computing facility and actually having more terminals than people). Recently, the population of terminals has shifted to greater and greater numbers of intelligent workstations, especially the IBM Personal Computer. The reasons for using workstations at the Research Center instead of terminals include increased productivity, easier personalization, and, especially in the research environment, the ease and low cost of attaching specialized equipment.

One major problem in migrating from "dumb" terminals to intelligent workstations is that many tasks are best performed on, or require the use of, a mainframe computer system. Mainframe computers offer capabilities and resources that are often unavailable on personal computers, such as high-speed computation and extremely large-scale storage for both computing use and archiving purposes. Mainframe computers also have an advantage in that they can access wide-area networking (including electronic mail facilities), shared data bases, and specialized devices, such as phototypesetting equipment that is much too costly to be provided on a personal system.

The Watson Research Center wanted to add the capabilities of a large mainframe system to the personal computer. One solution to this problem was a prototype project that eventually became a product named PC/VM Bond.

This paper discusses the factors that influenced the design decisions of PC/VM Bond, tells what those design decisions were, and then briefly describes the end product.

Design goals and assumptions for the prototype

Before design of the prototype began, goals for the prototype were clearly defined. After the personal computer user's needs had been studied, the goals were established and were to provide the following:

1. *A method of extending personal computer data storage using mainframe-based disk storage facilities.* The method should be easy (or possibly transparent) for the personal computer user to access and should provide for sharing of these data using the sharing capabilities of the mainframe operating system.
2. *Access to data created by or processed by host-based programs from a personal-computer-based program.* The user should not have to modify his personal-computer-based program to access these host data.
3. *Access to the communications network of the host.* This access should allow personal computer users to communicate with one another without greatly impacting personal computer usage.
4. *A mechanism for a personal-computer-based program to communicate* easily *with a host-based program, in a way familiar to personal computer application programmers.*

Just as important as the goals of the prototype were the assumptions. Assumptions about hardware and software requirements of the personal computer, the host system, and the intended audience of the prototype were given much thought.

The research group at the Watson Research Center assumed that the most widely used personal computer within IBM, the IBM Personal Computer (called IBM PC in this paper), would be the system from which to start. Other systems could be added to the list later. And, because the IBM Personal Computer Disk Operating System (PC DOS) is the most widely used software environment on the IBM PC, the prototype had to run concurrently with PC DOS. An added restriction required that the research group make no modifications to PC DOS, since the source code for PC DOS was not available.

The Virtual Machine/System Product (called VM in this paper) would be the operating system used for the mainframe computer. VM is very widely used, both inside and outside IBM, and, perhaps most important, is the most heavily used mainframe operating system at the Research Center.

VM also provided all of the functions needed to meet the goals set for the prototype. An attempt was made, however, during the design of the prototype, to restrict the dependency on VM where possible to allow future migration to other mainframe environments.

Just as important as the goals of the prototype were the assumptions.

With assumptions made about the personal computer and the host system, the research group focused its attention on the intended user. Because the goal of the prototype was to enhance the IBM PC, it was assumed that users would be familiar with their IBM PCs and PC DOS, but might be unfamiliar with the host system, VM. This made the goal of creating an invisible environment a primary task. *Thus, the main focus of the prototype was to give IBM PC users access to some of the services provided by VM without forcing users to leave the PC DOS environment.*

End-user functions of the prototype

Once the goals and assumptions were determined, it became possible to look at the problem at the end-user level. The research group decided the user should be able to

- Log on to the host from the PC DOS environment without having to learn how to use the host. The end user should only need to know the user identification and password for a host, and an automatic log-on procedure would do the rest.
- Create and use virtual disks from PC DOS. Although the virtual disk would actually be a fil

on the host, to the user the virtual disk would be used just as any other disk on the IBM PC.

- Convert host-formatted files to files contained in a PC DOS-formatted virtual disk (on the host) and vice versa.
- Send a message to another prototype (or host) user from the PC DOS prompt.
- Receive a message on PC DOS from another prototype (or host) user. The message should not disrupt the IBM PC session; the prototype user should be able to display the message when it is convenient. Displaying the message should not destroy data on the IBM PC screen, and the prototype should restore the screen to the state it was in before the message was displayed.
- Send commands to the host from the PC DOS environment and receive the results from the host back on PC DOS.
- Easily switch from PC DOS to using the IBM PC as a terminal on the host and, just as easily, switch back to a PC DOS environment.

Design considerations

After some preliminary design discussions and some experiments, the prototype design began to take form. This design required a number of components: a terminal emulator, a communications layer, device drivers, host-resident software, support utilities, and an automatic log-on facility.

Figure 1 shows the actual components of the prototype and how they relate to one another.

The terminal emulator. When discussion about the prototype began, the main concern was how the IBM PC should connect to and communicate with the mainframe computer. At the time, the most prevalent communications connection was a coaxial cable for the IBM 3270 display terminal. There were also experimental adapter cards available that could connect an IBM PC to a coaxial cable. Therefore, with the appropriate software, it was possible for the IBM PC to emulate a terminal using this connection. It was decided to make use of this terminal emulation capability by providing 3277 terminal emulation software as the communication mechanism for the prototype.

This mechanism had several advantages. Whenever a potential user wanted to convert to using an intelligent workstation, the existing IBM 3277 terminal could be disconnected and an IBM PC connected in its place. And, because the mainframe computer could recognize the IBM PC as a 3277 terminal, the operating system of the mainframe computer needed no reprogramming.

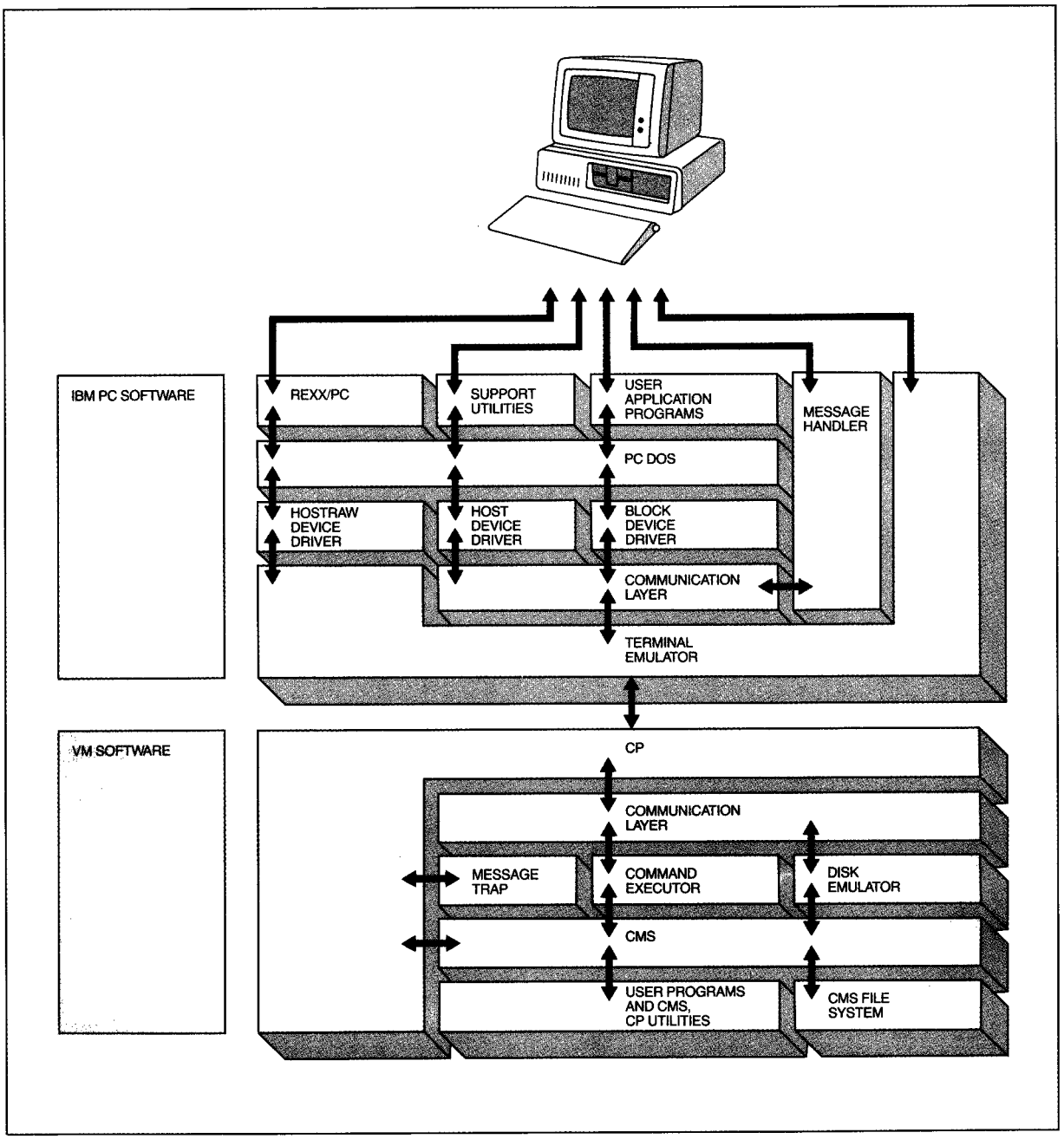
As an early version of the prototype was being completed, a hardware problem surfaced with the connection to the mainframe computer. The marketplace was shifting to IBM 3278 and other types of terminal emulation, so the adapter card was only produced in small quantities. It was also experimental and not destined to become a product. Fortunately, at that time, the 3278/79 Emulation Adapter Card for the IBM PC was announced as a product. This solved the hardware problem, but created a new problem. The 3277 terminal emulation software that was being used had become inadequate because it only worked with the obsolete adapter card.

The decision was made to write a new 3278 terminal emulator program providing additional features that were not available in some of its predecessors and tailoring it specifically to the needs of the prototype. Some of the features that were provided by the new terminal emulator were important, either because they provided improved human factors to the users of the emulator or because they provided capabilities needed by the rest of the prototype.

One important feature was a user-reconfigurable keyboard. There are many differences between the IBM PC keyboard and a 3278 keyboard. The IBM PC has fewer keys and a different keyboard layout. Experience with earlier emulators showed that user preferences for key mappings of IBM PC keys to 3278 functions varied widely. To meet the preferences of each user, the author of the terminal emulator found that he spent a significant amount of time creating "custom" patches for different users. It became obvious that a user-configurable keyboard mapping of IBM PC keys to 3278 functions would be ideal.

Another important feature of the new terminal emulator was a data-transport communications protocol. Other major portions of the system needed to move large blocks of data between IBM PC memory and mainframe memory. Because the terminal emulator already interfaced with the

Figure 1 The design of the prototype



3278/79 emulation adapter hardware, the emulator was the ideal place to provide the data communications protocol software to link to the mainframe computer.

The research group decided early that another desirable feature of the prototype would be the automatic log-on procedure to connect the user to the mainframe system from PC DOS. This feature

required a way for the prototype to *simulate* keystrokes to the mainframe computer and then to *read responses* from it. This was different from the data transport protocol, which assumed that a cooperative software package was already running on the mainframe computer. Again, since the emulator program was already providing the interface to the communications hardware, the software that simulated typing of keystrokes and reading of the screen was already available.

Finally, it was well known that many applications the user might perform were best done as a "native" mainframe computer user, that is, directly on the mainframe system. Therefore, the user might also want to use the workstation as if it were a 3278 display terminal. The emulator provided this capability.

Previous studies done at the Watson Research Center[1] showed that terminal response time has an enormous impact on productivity, so much so that specific goals are set for response time on the large computing systems. The terminal emulator had to provide suitable performance so that these response-time goals would still be attainable. This was a stringent requirement, because the goals were set when the most prevalent terminal in use was a 3277 display terminal. And, due to the dedicated hardware of the 3277, it operated at a very high speed.

The communications layer. A layered communications system provided the communication path between the IBM PC and VM. It was developed conceptually congruent to the International Organization for Standardization/Open Systems Interconnection (ISO/OSI) model.[2] To work efficiently, the communications layer was designed to satisfy several objectives:

1. The prototype would contain several subfunctions whose data had to be multiplexed over a single mainframe link. Ideally the subfunctions should have separate and distinct "streams" of information flow.
2. The prototype had originally been designed to use the experimental adapter card that was being phased out; therefore, the communications system had to isolate the rest of the prototype from the details of the specific hardware used for communication.
3. The communications devices used were not capable of transmitting binary data, but were limited to certain printable character sets. The communications system had to encode and decode the binary data passed to it from the rest of the system and allow direct communication of binary information.
4. Because future versions of the prototype might not be based on the 3270 coaxial cable connection, the research group tried to isolate the communications layer from the peculiarities of the 3270 architecture as well as the particular terminal emulator.
5. Although performance was not a prime consideration, especially since there were many factors involved, the group felt it reasonable to strive for a speed close to IBM PC diskette speed when accessing storage on the mainframe computer. If this goal could not be achieved, the research group had to ensure that the prototype software was not the bottleneck.

Some of the more traditional features of communications protocols were not needed by the communications system.

The prototype assumed the existence of a point-to-point logical connection between the IBM PC and the mainframe computer. This connection could be provided in a number of different ways; for example, the IBM PC could be connected to a control unit which was locally attached to the mainframe computer; the control unit could also be connected to the mainframe computer through Systems Network Architecture (SNA). The type of connection was not important to the prototype.

Because the underlying communications hardware could take care of the recoverable errors, there was little need for sophisticated error detection and recovery mechanisms. And, since almost all data traffic consisted of a single request followed by a single response, there was no need for more than an elementary flow control mechanism.

The device drivers and message handler. Very early in the design process, the group decided that the prototype should communicate with PC DOS through the device driver interfaces defined and published for PC DOS Version 2.0[3] and later versions. To PC DOS, the prototype would appear as a set of device drivers.

This design was chosen because it minimized the potential impact on user application programs and did not require any modifications to PC DOS. One of the ramifications of this design choice was that

The unit of storage on VM would be the virtual disk.

the unit of storage on VM would be the virtual disk (actually a Conversational Monitor System, or CMS,[4] formatted file).

Before implementing the prototype, programmers at the Watson Research Center discussed the merits of storing workstation data on the host either as virtual disk images or directly, as host-format files. Virtual disks had the advantages of easy implementation and very high transparency to IBM PC-based application programs. Host-format files had the advantage of easy migration between host- and IBM PC-based applications.

Because PC DOS already had an interface defined for device drivers and because almost all application programs used the PC DOS file system to access disk files, the virtual disk approach was compatible with most IBM PC-based software. The file-storage methods, on the other hand, required that the prototype convert VM file formats and character sets to IBM PC file formats and character sets dynamically when the I/O operations were requested from the IBM PC software. IBM PC file formats are quite different from those used on CMS. The CMS file system is record-oriented, with no delimiter characters between records, and it uses the EBCDIC (Extended Binary Coded Decimal Interchange Code) character set. PC DOS treats files as streams of characters, with no record structure imposed by the file system, and it uses the ASCII (American Standard Code for Information Interchange) character set.

Sequential reading and writing of files containing readable text is relatively easy using a file-oriented storage method. But files that deal with random access or that contain a mixture of text and binary data (like many of the spreadsheet programs used on the IBM PC) are extremely difficult to read and write correctly.

Another problem with the file-oriented approach was that the defined PC DOS driver interface was at a low level: It only dealt with sectors on a disk, not file opens and closes. To obtain the necessary information from the application program to map the DOS I/O requests to the appropriate operations on the host system, it would be necessary either to modify PC DOS or to trap all PC DOS calls and filter out and redirect those that were destined for host-based files.

Ultimately, the virtual disk strategy was chosen, primarily because it was possible to implement it using the PC DOS device driver interface without extensive modifications to, or completely bypassing, PC DOS. The decision has proved to be a good one since most PC applications can use virtual disks provided by the prototype in the same way as they use the real diskettes and fixed disks on the PC itself.

Once the research group decided to use the PC DOS device driver interfaces, the need arose for three separate drivers. One device driver, called HOSTRAW, is used during automatic log-on to simulate an operator at a 3278 Model 2 terminal. It is unique among the three device drivers in that it does not require any corresponding communications software to be running at the host.

After the log-on procedure is completed and the host-resident software of the prototype is running on the mainframe computer, a second device driver, called HOST, sends commands to VM and receives the results of these commands back on the user's IBM PC. HOST uses the communications layer to communicate with software in VM that executes the commands. The output of the commands is then redirected through the communications layer back to the IBM PC.

PC DOS calls the third device driver, the *block* device driver, whenever a user application program (or PC DOS command) requests a PC DOS operation on one of the virtual disks. The block device driver communicates through the communications layer with the host-resident software on VM. (The

host-resident software is a part of the prototype that resides within VM. It performs disk functions on the virtual disks and sends the results back through the communications layer to the PC.)

The message window handler is the only remaining major function of the prototype not handled by a device driver. The message window handler displays message windows on the PC screen. It is an interrupt-handling routine that is invoked periodically by a timer interrupt. When invoked, it checks to see if any messages are waiting to be displayed, and if so, displays them when the appropriate keys are pressed. It restores to the display screen its previous contents when the same keys are pressed again.

Host-resident software. The prototype needed many functions that could not be performed from PC DOS. The functions had to be performed within VM.

From VM, the host-resident part of the prototype had to trap all Control Program (CP)[5] messages from CP and redirect them to the appropriate part of the IBM PC software. Fortunately, VM/SP Release 3.0 provided a new feature, the Virtual Machine Communications Facility (VMCF), which allowed messages and command output to be trapped and processed by the host-resident software.[6]

Another function required that the host-resident part of the prototype accept requests from the IBM PC to perform VM commands and direct the console output from these commands back to the IBM PC.

Because CMS does not support multitasking, the most difficult problem in providing this function was running CMS (or CP) commands under the direction of the prototype host-resident software so that the output could be directed back to the IBM PC. Essentially, the prototype host-resident software had to become part of CMS itself, out of the way of executing programs, and yet still be able to trap the console output and direct it to the IBM PC.

CMS and CP mechanisms allowed this. CMS provided the NUCXLOAD feature enabling a program to become a resident extension of the CMS nucleus. Thus, the prototype host-resident software no longer occupied the dynamically allocated memory area into which other programs would be loaded. CMS considered the host-resident software an extension of the nucleus.

Finally, the prototype had to simulate all of the functions of a diskette (or fixed disk) controller for an image of an IBM PC disk. The simulation of a real device had to be at the level of detail appropriate for the PC DOS driver interface, not necessarily at the level of a real disk controller.

One of the problems encountered later in the development and testing of the prototype was the way CMS handled updates to a virtual disk file. Whenever an IBM PC program changed a sector on a virtual disk, CMS was requested to rewrite a 512-byte sector image in the virtual disk file. CMS could allocate a completely new block on its disk to contain the new sector image, discarding the old block only when the file was closed. Thus, an IBM PC program that updated a large number of disk sectors could run out of disk space on the CMS minidisk, even though no new information was being added and the size of the virtual disk remained constant. Fortunately, Release 3.0 of VM included a new "update in place" feature in CMS, eliminating the problem.

Support utilities. The prototype utilities were created to provide additional PC DOS "commands." These commands had to perform the special functions for the prototype: create a virtual disk, activate a virtual disk, control the display of VM messages, and a few others that will be discussed in greater detail next. The utilities accept parameters on the command line (not via a menu), and the utilities and their parameters can be integrated into PC DOS batch files. With this, the user can build superset commands based on the fundamental prototype commands.

Sending commands to VM for processing. One utility was the VM command. Using this command, the user could send line-mode commands to VM and receive back line-mode responses without leaving the PC DOS environment. (VM line-mode commands are those commands that produce output a line at a time; that is, line-mode commands do not produce any full-screen data or screens.) The user could elect to have the responses queued for later review. With this option, the user could perform a long-running task on VM while PC DOS was free to perform another task concurrently.

Transferring files from virtual disks to VM. After the decision was made to store the virtual disk in absolute PC DOS disk image format, the need arose for a means to allow host applications to access files on the virtual disk. This included the desire to support the hierarchical file structure of the virtual disk and to insert host files into the virtual disk and extract files from it as host files.

A utility called PCDISK was created to run within VM.

A utility called PCDISK was created to run within VM. Using PCDISK, the directory structure and file allocation tables of the virtual disk could be examined or modified to perform the desired extracting or inserting.

Because the record-oriented file system of CMS differs from the byte-oriented file system of PC DOS, mapping takes place during the insertion and extraction processes, depending on whether the file is text or binary data. The text mapping requires that PC DOS carriage return characters be interpreted as end-of-record and that PC DOS end-of-file characters be interpreted as end-of-file. A print mapping was provided in which PC DOS files could be extracted with carriage control information placed in column 1 of the output file.

Automatic log-on procedure. One of the design objectives was for the IBM PC-resident software to establish a connection to the host-resident software in a manner that would be invisible to the user. This objective implied some sort of automatic log-on program that could simulate typing the necessary keys to log on to a host user's identification and start up the host-resident software.

The research group realized early that a hard-coded program would be too inflexible to accommodate the anticipated variety of host connection procedures. Therefore, the group decided to write a simple interpreter that could follow a site-specific script to perform a log-on sequence. Software was provided to send keystrokes to the host and then inspect the resulting screen image. The flow of control could then be altered on the basis of the observed screen image.

By the time enough capabilities had been placed in the interpreter so that a log-on script could be written, a complete but entirely nonstandard language had been designed.

Since a widely used interpreted language called Restructured Extended Executor (REXX)[7] was available on the host (but not on the IBM PC), the group decided to set the other language aside and write an IBM PC version of REXX that could perform the automatic log-on procedure. This programming language became known as REXX/PC. This version had several benefits. The need to define "yet another language" was avoided, providing a portable version of a language heretofore available only on VM. The REXX/PC language extended the PC DOS batch facility, and a common language for tying together user-written host- and IBM PC-based programs was made available.

The resulting product

PC/VM Bond, the resulting product, was created cooperatively by groups within the IBM Endicott Laboratory and the Thomas J. Watson Research Center. PC/VM Bond contains two separate products, VM Bond and PC Bond. VM Bond runs on the IBM Personal Computer and PC Bond (referred to earlier as the host-resident software) runs on the host. With PC/VM Bond, users of the IBM Personal Computer, Personal Computer XT, and Personal Computer XT/370 can

- Use disk storage on VM from PC DOS.
- Send messages to and receive them from other PC users using PC/VM Bond or other VM users.
- Emulate a 3278 Model 2 terminal connected to VM.
- Issue commands to VM from within PC DOS.
- Write programs in the REXX/PC language on PC DOS similar to the REXX language on VM.
- Develop cooperative applications that make use of both VM and PC DOS.

Using host storage from PC DOS. Using PC/VM Bond functions, users create *virtual disks* from PC

DOS that are actually files within the users' CMS disk space. When active, these virtual disks act as fully functional disk storage drives: The PC treats virtual disks like active diskette drives or fixed disk drives.

Users can create any number of virtual disks depending on the amount of CMS storage that each user has available. Up to eight virtual disks can be active at any time. The size of each virtual disk, set when the disk is created, ranges from a practical limit of 2 kilobytes to over 30 megabytes.

PC/VM Bond contains two separate products.

Besides the vast storage available from virtual disks on an IBM PC, users can share virtual disks with other PC/VM Bond users. And users familiar with VM can insert CMS files into virtual disks and extract files from virtual disks as CMS files. This procedure allows users to upload and download files between VM and the IBM PC without worrying about converting the files to the appropriate host or IBM PC format.

This file manipulation is a powerful function that can be used for a number of purposes:

- Taking VM-generated data and processing those data using IBM PC-based programs and vice versa.
- Printing IBM PC-based programs and data on high-speed, high-quality, VM-based printing devices.
- Developing cooperative applications between PC DOS and VM. An example of a cooperative application is shown later in this paper under the subheading "Using the REXX/PC Language."

Sending and receiving messages. By using a PC/VM Bond command, PC DOS users can send messages to other PC/VM Bond users or other VM users, either on the same VM system or on a different VM system connected by a network.

The PC/VM Bond function sends a CMS TELL command (not to be confused with the VM note facility) to VM for processing. PC/VM Bond users can communicate with one another as well as with "regular" VM users without leaving the PC DOS environment.

Messages sent to a PC/VM Bond user are held within the user's VM system until PC/VM Bond can accept them on the IBM PC. Once a message is transferred from VM to the IBM PC, a three-tone beep alerts the user that at least one message is waiting to be displayed. To display a message, the user presses a set of keys that causes a message window to appear displaying the message. To clear the screen of the message, the user presses the same set of keys once again.

Emulating a terminal. PC/VM Bond users can use VM fully by having their IBM PCs emulate 3278 Model 2 terminals. Users simply press two keys simultaneously to switch from PC DOS to terminal emulation and the same two keys again to switch back.

PC/VM Bond supplies a program that allows users to modify the distributed keyboard and color assignments. Users can reassign any of the 3278 functions and characters to other keys on the IBM PC keyboard. They can also reassign the way that color is displayed for each possible 3278 field attribute.

Issuing VM commands from PC DOS. Users can enter a certain subset of VM commands (those that do not produce full screen output) from PC DOS and receive the results back on PC DOS either directly from the PC DOS prompt or from within a program. If a VM command is entered from the PC DOS prompt, a user can also request that the results of the VM command not be returned immediately, thus allowing the IBM PC and the mainframe computer to perform user tasks concurrently. In this case, the results are held until the user asks for the results to be displayed.

For example, from PC DOS, a user could send a command to VM asking for the current time, which would be displayed on the PC DOS screen. From PC DOS, a user could also ask VM to compile a program and then continue to work on PC DOS.

Later, when VM had finished processing the command, the user could ask VM for the results of the compile.

Within the discussion of REXX/PC which follows is an illustration of how a user can send commands from a program running on PC DOS to VM (through the HOST device driver). Another language such as BASIC, C, or Pascal could have been used, but for purposes of illustration REXX/PC was chosen.

Using the REXX/PC language. REXX/PC, a programming language for the IBM PC, closely resembles the REXX language available on VM. REXX/PC can be used to supplement or to replace the batch command executor of the IBM PC. It provides a language common to both PC DOS and VM for creating new applications from existing host or IBM PC programs.

REXX/PC is primarily intended to be an exec language. Its data and control structures are designed to allow convenient argument processing, program execution, and return code analysis. Thus REXX/PC is often used to replace several small programs with a larger application.

The REXX/PC language is somewhat different from many other languages because it is string-oriented and typeless. The designation "typeless" means that data objects need not be declared before use, and, for most operations, no distinction is made between numeric and textual data. This means that there are few fundamental concepts to learn and also makes possible some interesting operations that would be difficult to do in a typed language.

The treatment of data objects is designed to make string manipulation and command execution especially easy. The rules for data objects are as follows:

- Until an object is assigned a value, its value is its name.
- The value of anything in quotes is the text between the quotes.
- The value of an expression formed by placing names and quoted strings adjacent to one another on a line is the concatenation of the values of those names and quoted strings.
- A line that does not begin with a keyword is evaluated according to the preceding rules and passed to the operating system for execution as if it had been typed by the user at the command line.

In addition to the standard arithmetic operators (addition, subtraction, multiplication, integer division, and remainder division), the language supplies a powerful string-parsing operator for dissecting strings into words (delimited by blanks) or fields (delimited by substrings, column numbers, or a combination). Then new variables can be created from the pieces.

Finally, a naming convention in the language allows more complicated objects such as trees, dictionaries, and arrays to be built from simple objects. This naming convention employs the use of "subscript" names to distinguish members of a group, all of which share a common "stem" name. The subscripts need not be numeric (any string or expression can be used). When a name contains subscripts, REXX/PC automatically evaluates the subscripts and then looks up the value of the resulting name.

In addition to the string-oriented aspects just mentioned, REXX/PC provides the control structures common to most programming languages:

- if then
- if then else
- subroutine calls
- function calls
- do ... end
- do forever ... end
- do count ... end
- do i=start to stop by step ... end

As with most programming languages, functions and subroutines can receive arguments, manipulate private variables, and access the variables of each caller. In addition, functions return a value to their caller.

One aspect of a function or subroutine call that is unique to REXX/PC is that a function or subroutine can call a routine located elsewhere in the system, for example, on another disk or path. If the named routine is not in the current program, REXX/PC searches the disk for a REXX/PC program whose name is the same as the routine, then loads and executes it. Control returns to the caller when

Figure 2 Example of a REXX/PC program

```
/*-------------------------------------------------------*
 *   A REXX/PC program to synchronize the PC's           *
 *   clock/calendar with that of the host                *
 *-------------------------------------------------------*/

/* Issue a host command to display the date and time.
 * It will be returned in the following format:
 *
 *  TIME IS 14:27:13 EST THURSDAY 02/28/85
 *  CONNECT= 05:27:43 VIRTCPU= 000:05.95 TOTCPU= 000:10.35
 *  R; T=0.01/0.01 14:27:13
 *
 */
call write 'host', 'CP QUERY TIME', 'eol'

/* Read back 1st line of response, extract the desired words */
parse value read('host') with . . time . day date .

/* Read back (and throw away) remaining two lines of response */
do 2
   call read 'host'
end

/* Issue PC commands to set the date and time */
'date' date
'time' time

/* Display the results */
say 'Today is' day date
say 'The time is' time
```

the routine exits. This allows complicated execs to be built from separately tested components.

To aid debugging, REXX/PC can print an execution trace with indentation to match the subroutine nesting level. Arguments and return values can be displayed when entering or returning from a procedure. A pattern can be set to step through a program during the execution of a trace. The current values of all private and global program variables can be displayed at any point.

The interpreter requires about 30K of storage and up to 64K (default = 24K) for data. REXX/PC explicitly creates and destroys objects as needed. To perform arithmetic, REXX/PC converts strings into numeric format, performs the arithmetic operation, and then converts the result back into a string.

The interpreter operates in two phases. In the first phase the source program is tokenized. The tokenized program is converted into an intermediate form in which keywords are represented by integers and literals are collected into a common area (compressing out duplicates). In the second phase the tokenized program is executed.

The results of the first phase can be saved to disk if the user so requests. This allows the program to load faster the next time it is executed. The tokens are conveniently saved by appending them to the end of the original source file, after the end-of-file mark. Since most programs ignore file characters following an end-of-file mark, the tokenized portion is invisible to most programs, but unlike other languages, REXX/PC can still find it. Furthermore, the program automatically becomes "detokenized" the next time it is edited.

Figure 2 shows an example of a REXX/PC program that interacts with VM and PC DOS. The program sets the date and time on the IBM PC to the date and time of the host VM system.

Although the fundamentals of REXX/PC can be learned in a few hours, the language is rich enough to create functions that would be difficult or time-consuming to code and test using compiled languages. In addition, REXX/PC introduces programmers to the CMS command interpreter and can help them tailor IBM PC and VM environments without their having to learn about the system compilers, assemblers, linkers, and debuggers.

In the future

A number of interesting possibilities are being discussed at the Research Center for possible future application using the tools provided by the prototype. These include

- An editor that consists of a highly interactive front end on the IBM PC, cooperating with a data-storage and editing "engine" on the host
- Automatic mail systems that direct network mail and messages directly to the user's IBM PC
- Knowledge-based systems that attempt sophisticated error recovery and retry operations in controlling the operation of a large-scale computing center

Summary

The prototype and some of its components, such as the terminal emulator and REXX/PC, have become widely used within the Watson Research Center and throughout IBM.

A number of IBM locations are using the prototype or PC/VM Bond to perform a wide variety of functions which include

- Turning an IBM PC into a user-friendly front end to VM including a menu-driven interface, pop-up message windows, and access to VM spool files
- Using an IBM PC to generate and display high-resolution graphics images, and using the host to perform computation-intensive calculations on the graphics data; the graphics data are then displayed on the IBM PC
- Using an IBM PC to control laboratory instruments and pass the results to a host program for data analysis
- Gathering statistics about VM system response time performance using an IBM PC to simulate user interaction with a host
- Controlling a large-scale mainframe computer center automatically, using an IBM PC to intercept operator console messages, filter out unnecessary information, and respond to certain messages automatically
- Sharing and updating a workbench of internally developed user tools and data among other IBM PC users
- Backing up IBM PC data and programs on a host system

Acknowledgments

The authors wish to acknowledge the contributions of Matthew Zekauskas, Dave Chess, and Dave Levine, whose work was instrumental in completing the prototype, as well as Mark Giampapa, whose applications made use of the prototype. Others to be thanked are the entire Endicott team, who turned a prototype into a product; Gerry Waldbaum, who managed the prototype project; Harry Benas, who was the product manager for PC/VM Bond; and Dick Kerr, whose literary expertise was of great help in completing this paper.

Cited references and notes

1. W. J. Doherty and R. P. Kelisky, "Managing VM/CMS systems for user effectiveness," *IBM Systems Journal* **18**, No. 1, 143–163 (1979).
2. *Data Processing—Open Systems Interconnection—Basic Reference Model*, Draft Proposal ISO/DP 7498, International Organization for Standardization, Geneva (February 1982).
3. *IBM Personal Computer Disk Operating System Technical Reference*, IBM Corporation; available through IBM branch offices and authorized IBM Personal Computer dealers.
4. The Conversational Monitor System (CMS) is the part of VM that manages its users and their file systems, etc. Details are in *VM/370: CMS User's Guide*, GC20-1819, IBM Corporation; available through IBM branch offices.
5. The Control Program (CP) is the part of VM that intercedes between CMS and the hardware system.
6. *Virtual Machine/System Product System Programmer's Guide*, SC19-6203, IBM Corporation; available through IBM branch offices.
7. M. F. Cowlishaw, "The design of the REXX language," *IBM Systems Journal* **23**, No. 4, 326–335 (1984).

General references

PC/VM Bond User's Guide, No. 6317007, IBM Corporation; available through IBM branch offices.

PC/VM Bond Programmer's Guide, SH24-5097, IBM Corporation; available through IBM branch offices.

PC Bond (the host-resident product), No. 564-298; available through IBM branch offices.

VM Bond (the product that runs on the IBM PC), No. 6467022; available through IBM branch offices.

IBM Personal Computer Disk Operating System, IBM Corporation; available through IBM branch offices and authorized IBM Personal Computer dealers.

Virtual Machine/System Product Introduction, GC19-6200, IBM Corporation; available through IBM branch offices.

Introduction to the IBM 3270 Information Display System, GA27-2739, IBM Corporation; available through IBM branch offices.

Jeff K. Kravitz *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Before joining IBM, Mr. Kravitz worked for a number of years for several companies as a designer and implementer of operating systems. Upon joining the IBM Research Division in 1983, he became the technical project leader of the prototype of the PC/VM Bond system. He was responsible for much of the overall design of the prototype and the implementation of the terminal emulator and the communications layer. He is currently working in the area of gateways between large computer systems and local area networks. Mr. Kravitz obtained his bachelor's degree in mathematics from Queens College of the City University of New York.

Derek Lieber *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Lieber joined IBM in 1983 and currently works in the Advanced Workstation Projects group. He is a 1975 graduate of Lebanon Valley College with a B.S. in physics; his interests are interpreted languages and interactive user interfaces.

Frederick H. Robbins *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Robbins, a staff programmer, has been with IBM since graduating with a B.S. in electrical engineering from Clemson University in 1968. After serving in the U.S. Army, he worked at IBM Poughkeepsie until 1981 on a variety of assignments, including microcode, applications programming, and consulting on the 3080 and 3090 processor development projects. For the last four years, Mr. Robbins has been at the Thomas J. Watson Research Center, first in Signature Verification and then with Advanced Workstation Projects, where he was a major contributor to the prototype, developing the PC-based code for the virtual disk emulation. He is now a PC consultant at Yorktown, while continuing to develop workstation applications.

Julie M. Palermo *IBM Systems Technology Division, P.O. Box 6, Endicott, New York 13760.* Ms. Palermo joined IBM in 1982. She has been involved in Information Development, working on IBM Personal Computer (and VM/SP-related) products and documentation. Ms. Palermo is currently an associate information developer working on PC/VM Bond. She has a B.A. in mathematics from the State University of New York at Potsdam.

Reprint Order No. G321-5266.