vision system can allow everyone to see the displayed information at once, although only one person can use the keyboard at a time.

The above examples point out that this solution is often either expensive or impossible. Travel is expensive in both money and time, and the expense involved in getting people together can easily be more than the benefit to be gained. Even when the people can be brought together physically, many I/O devices will not conveniently allow more than one person access; this is particularly true of input devices, such as keyboards or tablets.

In today's workstations, applications are designed to interact with a single user.

A solution. An ideal solution to this problem would enable each person involved to interact fully with the relevant computing environment from his or her own workstation, in as transparent a manner as possible, and with as little penalty (in terms of speed, reliability, and the like) as possible.

To accomplish this, at least the following elements are required:

1. A means of capturing relevant output from the computing environment for forwarding to the remote participants
2. A means of providing the computing environment with input received from remote participants in such a way that the environment will treat the input as though it came from the normal local input devices
3. A communication method to connect the workstations

This conception of the problem and its solution is based on the assumption that the computing environment with which the participants need to interact does not "know" about the remote participants. This is typically the case in today's workstations; applications are designed to interact with a single user, and that user is assumed to be physically located at the workstation itself and to be using the standard input and output devices of the workstation.

The requirement for a communication method between the workstations is the easiest to satisfy; there are numerous methods, and designs for methods, of communicating information between workstations. The only special requirement in the present case is that the communications method must be able to operate "behind the scenes" at the same time that the application of interest is executing in one of the workstations. This requirement implies that at least a primitive (interrupt-driven) type of multiprocessing must be available. Capturing of output and provision of input are more difficult issues; they will be taken up in the next section.

Systems have been implemented that address this sort of problem in various environments. IBM's recently announced Cooperative Viewing Facility,[2] for instance, uses Virtual Machine (VM) Logical Device Support for elements 1 and 2 and already-existing terminal-to-host connections for element 3. It can thus support workstations connected as terminals to a VM mainframe host. The prototype system described later in this paper (the "coupler") is being used to study possible implementations of this sort of solution in a microcomputer workstation environment.

Approaches. One of the functions of an operating system is to provide, in a well-defined and structured manner, a way for application programs to perform I/O operations with user devices such as keyboards and display screens. This has an important implication for our problem. Somewhere between the application code and the devices, there are typically places where "the path is narrow"; where, that is, some vector or entry point will always be used to do input or output operations using the operating system interfaces. A program that accomplishes objectives 1 and 2 can step in at the narrow places in the path and, for instance, intercept all writes to the display and all queries to the keyboard.

By seeing every attempt by a program to write to the display, a program doing this sort of interception can forward the appropriate information to the remote workstations, where a simple application program can take the information and update the remote displays accordingly. Similarly, the programs

at the remote workstations can send any input that they receive to the intercepting program, which can provide the input in the usual way when the application asks for it. If all relevant narrow paths are identified and intercepted, and all applications use these paths for their input and output, the full transparency we want can theoretically be achieved.

The actual picture is not as rosy. Particularly in small, self-contained workstations, applications are not constrained to using the operating system interfaces to do device I/O operations. For reasons of speed, or just because of sloppy programming, applications often read and write directly from and to memory locations and hardware ports connected to I/O devices, rather than using the standard interfaces. This means that the intercepting program must become more complex, and that in some cases the software cannot provide full transparency, regardless of cost.

Communications issues are another complicating factor. They will not be discussed here at any length, but it is good to keep in mind that even a relatively reliable communications medium may occasionally lose a message and thus cause the remote workstations to become in some sense out of step with the actual display. Since it is generally anticipated that the people at the various workstations will be in verbal communication, the remote participants should be able to notice this and request that a synchronization function be performed. This function might be as simple as clearing the display screen, or it might require special action on the part of the interception program.

With these considerations in mind, we can outline the requirements for a system to address the problem:

- On one of the workstations, there must be a means by which the interception and communication programs can run in the background, in parallel with the application of interest. This requirement implies that at least a primitive level of multiprocessing must be available on the workstation.
- The interception routines must be able to monitor the "narrow places" in the relevant I/O paths, without interfering materially with their normal function.
- All relevant I/O operations by the application of interest must be done through I/O paths that the interception routines are monitoring. To the extent that unmonitored paths exist, some I/O device will be unavailable, or only partially available, to the remote participants.
- The communications program must be able to communicate information between the workstations at a rate acceptable to the participants.

The remainder of this paper presents a prototype solution to the problem.

A prototype solution

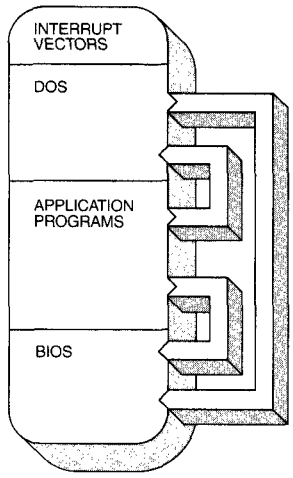
A prototype system has been developed at IBM's Thomas J. Watson Research Center to study ways of implementing a solution to the problem of having a simultaneous computing environment at separate, remote workstations. This section describes the workstations used by the prototype, the nature of the I/O paths intercepted, the types of information exchanged between the workstations, and the communications method used. The subsequent sections describe some uses to which the system has been, or might be, put, as well as some possible future extensions. For brevity, the prototype system is referred to as "the coupler" in what follows.

The workstations. The coupler was developed to run on workstations in the IBM Personal Computer (PC) family[3] (the Personal Computer, Personal Computer XT, and PCjr) under the IBM Personal Computer Disk Operating System (DOS).[4] The choice of operating environment was largely pragmatic; it is the one used by most of the workstation (as opposed to terminal) users at the Research Center. It proved to be a fortunate choice, though, in that the narrow places in the I/O paths were few and readily intercepted. As will be discussed later, interception of the appropriate paths did not completely solve the problem.

Narrow paths. There are two levels of I/O interface that are specified by the architecture in the IBM PC DOS environment. The higher level consists of calls to DOS proper. This interface provides simple interfaces with somewhat limited function. There is a call to write a character to the display, for instance, but no way to clear the display, or to determine where on the screen the next character will appear.

The lower level (which is called by the higher one) is known as the BIOS (for Basic Input/Output System) interface. It provides a more detailed control of the display, and more generalized access to the keyboard. Since calls to the DOS interface result in calls to the BIOS, intercepting the BIOS calls takes care of both levels.

Figure 1 Service calls in normal workstation operation



Calls to the BIOS are made through a so-called "interrupt" instruction, which results in control being transferred to an entry point defined by one of a set of vectors at the bottom of the memory of the PC. (See Figure 1, where it can be seen that application programs may call the operating system, which resides in low memory, or the I/O subsystem, which is in read-only memory (ROM) in high memory. The operating system also calls the I/O subsystem.) Intercepting BIOS calls, then, consists simply of recording the old contents of the appropriate vectors, and then substituting the addresses of the proper entry points in the interception code.

At the "host" end of the connection, the code involved consists of an interceptor for BIOS display output calls, and one for BIOS keyboard input calls. These interceptors are installed in main memory and made "resident" by the appropriate operating system requests; they remain in memory while other applications are run, until memory is cleared by a system reset, or power down. (See Figure 2. At the host workstation, some calls that would normally go to the I/O subsystem are diverted through the coupler code first by changing the contents of certain "interrupt" vectors. The coupler code in the host communicates with the terminal workstation code, which runs as a standard application under DOS.) At the "terminal," or remote, end of the connection, the code consists of a single application program, running under the operating system in the more usual way. This program simply interprets messages coming in from the interceptor code in the host, updates its display accordingly, and informs the host about any keystrokes entered by the user at the remote workstation.

Many of the service calls to the BIOS concern devices that the coupler is not concerned with—the disk drives of the workstation, the time of day clock, and the like. Only the following calls to the BIOS are intercepted as being of interest to the coupler:

- Change the shape of the hardware cursor
- Change the cursor position
- Scroll the screen up
- Scroll the screen down
- Write a character at the current cursor position
- Determine if a character is ready to be read from the keyboard
- Read a character from the keyboard

Semantics. Except for some protocol information used to establish the logical connection in the first place, and to terminate it when necessary, the programs in the "host" and "terminal" workstations need to exchange only a few types of messages. The terminal program must be able to inform the host about relevant input received (in the current prototype, this consists only of keystrokes), and the host must be able to inform the terminal about any changes to be made to the display.

In the current implementation, the following types of information are defined in host-to-terminal communication:

- Request for initiation—Sent by the host to set up the link with the terminal workstation.
- Request for shutdown—Sent by the host to request termination of the session.
- BIOS service call—Notification to the terminal that the given service call, with given parameters, has been executed on the host. If all applications utilized the BIOS for all display output (see the remarks below), this information would completely determine the screen contents.
- Changed display information—There are several variants of this message: one gives a position on the screen and a single character to appear there, one gives a starting position and a number of characters, and a third gives a starting position, a single character, and a repeat count specifying how many times that character is to appear. By balancing the use of these variants, the performance of the communications link can be slightly improved.

- Display cursor position—This is used by the host to notify the terminal that the position of the hardware cursor of the display has changed.

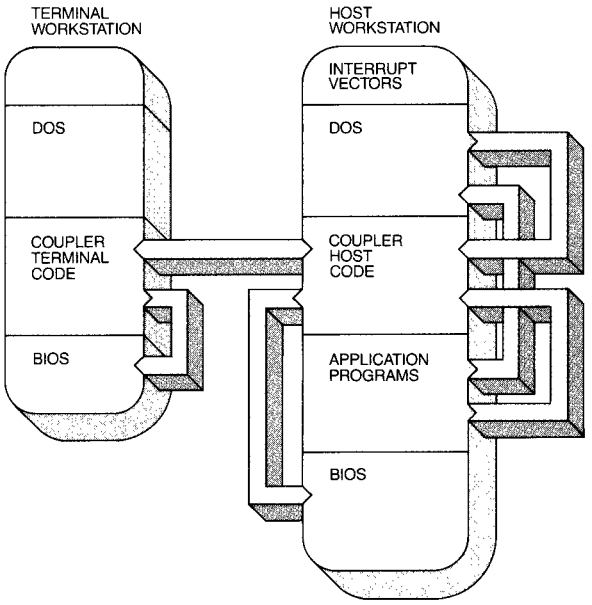
The following message types are defined for terminal-to-host communications:

- Grant initiation—Used as a positive acknowledgment of the request-for-initiation message.
- Acknowledge shutdown—Used as a positive acknowledgment of the request-for-shutdown message.
- Keystroke—Notification to the host that the given key on the terminal has been struck.

Support for more complex input and output devices would require the addition of more types of messages between the workstations.

Note that the details of the communication protocol (requiring the host end to request initiation and termination of the session) were more or less arbitrary, and the same ends could have been accomplished with the reverse protocol, or probably with no particular protocol at all.

Figure 2 Operation with coupler installed



Communication. The prototype implementation of the coupler uses asynchronous communications to connect the workstations. On one hand, this method is generally available, given the growing number of workstations that have built-in or optional modems and an easy connection to a telephone system. On the other hand, it is neither very fast nor particularly reliable.

The communications code was implemented as a set of subroutines so as to require the interception code to know as little about the communication method as possible. Modifying the prototype to use another medium (a local area network, for instance) would primarily require changes only to these subroutines.

User protocols. As the prototype is currently set up, the physical communication link between the workstations is established first (through a direct cable connection or through modems and telephones). Then the terminal workstation program is invoked to wait for an initiation request from the host. When the host program is run, it sends the initiation request, waits for a grant of initiation in reply, and then installs the communication and interception programs and returns control to the operating system. From then on, the workstations are coupled together; anything typed on either keyboard is made available to the BIOS keyboard input routines, and anything displayed on the host screen is reflected to the terminal for display there.

If the telephone system is being used to connect the workstations and only one telephone is available at each workstation, it may be difficult for the participants to communicate directly. If the application being run has a free-form input area of some kind (such as an editor data area, or a general command line), one user can just enter a message from the keyboard, and the other user will see it when the application displays it on the screen. In menu-driven systems, or other environments without a free-form input area, this type of operation will not be possible.

Since it will in general be desirable (and even necessary) for the participants to communicate directly, an extension to the coupler that would allow an immediate message to be passed from one workstation to the other, independent of the active application, might be useful. However, such a facility would have the drawback of being dependent on the correct functioning of the workstation communications link (one of the reasons the participants might want to communicate directly would be to ask "Has the link gone down?") and would involve some kind of violation of transparency (if sending a message in-

volved entering a particular keystroke, for instance, that keystroke could no longer be passed to applications).

Another possible means of direct communication during a coupler session is a second telephone line. Workstations used heavily for communication may already have two telephone lines, so that the user can be available for incoming calls while using the workstation as a mainframe terminal, for instance. The point to be borne in mind is that the connection of computing environments may not be enough; if the users are physically remote from one another, they will want a means of communicating directly, rather than through the environment. Such communication may be done either through the workstation connection or through a separate medium.

The coupler may be used to present information in a conferencing or teaching environment.

When the coupler session is over, one of the users invokes the host program with special arguments, and the host workstation sends a request for shutdown. The terminal responds with an acknowledgment, and returns control to the operating system. The host program then disables the interception routines and returns control to the operating system in the host.

There is also an "emergency exit" key in the terminal, which may be used to return control to the operating system at the terminal end in case, for instance, the communications link fails. This key involves a slight violation of transparency but was found to be necessary during development and testing.

Uses

As the prototype was being developed, people were continually thinking of more situations in which it could be useful. The summary of situations given here is not intended to be exhaustive by any means.

Remote troubleshooting. Customers and other personal computer users often have difficulties that seem to require the help of software service personnel. With present methods, this requires either travel on the part of the service person (which is expensive and time-consuming), "talk-throughs" over the telephone (which do not generally work very well), or the mailing of printed output such as dumps (which is very slow and inflexible). With the coupler, the customer's PC can serve as the host end of the connection and the service person's as the terminal end. Since the machines may be connected over telephone lines, the service person will not have to leave the service location.

Remote demonstrations. Personal Computer software is usually demonstrated either by travel (the "road show" technique) or by mailing diskettes and documentation. The first method is expensive, time-consuming, and often not cost-effective. The latter method is unreliable (diskettes are damaged in shipment), limited (potential customers cannot ask questions as they occur to them and may be unable to use the software correctly without help), and sometimes impossible (proprietary software should not be shipped).

With the solution outlined here, demonstrations can be conducted under direct control of the program owners, without allowing the software to leave the shop and without any travel expense. The demonstrator's PC acts as the host machine, and that of the audience serves as the terminal. The owner may conduct the demonstration, and the audience can try the software out themselves, almost as if they were in the same room. Thus, most of the benefit of travel is obtained, without the expense and inconvenience of actual travel or the danger of software shipment.

Presentations, conferencing, and education. The coupler may be used to present information in a conferencing or teaching environment. The host PC is used by the person presenting the information (the presenter or teacher), and the other PC is used by the students or conference participants. With a voice connection active at the same time as the coupler connection, the presenter can cause information (perhaps in a viewgraph-like format) to appear on the screen of the terminal PC, watch it on the host PC, and discuss the information, all without leaving

his office. With the appropriate conferencing software running in the host PC, those at the terminal PC could also enter information (questions or feedback) to be displayed on both screens.

Mainframe remote shared terminal support. Individuals, such as service personnel, who are not themselves authorized to use a certain mainframe computer system must occasionally have access to that system for short periods of time, under supervision from system personnel, to aid in problem determination or similar activities. With conventional methods, this access must be available either by travel or by temporarily authorizing the individuals to access the mainframe remotely. The latter method may be either impossible (many mainframes have no provision for remote access) or an unacceptable security exposure.

The speed of the communications link is a limiting factor.

If software is available (and it generally is) to allow an IBM Personal Computer to emulate a terminal to the mainframe, the coupler can be used to allow completely supervised remote access to the mainframe. The host PC runs terminal emulation software and connects to the mainframe as a terminal. The service person's PC acts as the terminal end of the connection. In this environment, the service person can issue commands to the mainframe through the terminal PC without leaving the service location, and the owners of the mainframe system can control everything that is done by watching and entering commands on the host PC.

Note that access to the mainframe via the coupler does *not* depend on the mainframe having remote access hardware; it merely requires that the host PC have access to the mainframe.

Resource sharing. The coupler also turns out to be useful in a situation for which it was not specifically designed. It can be used in an environment containing workstations with various levels of hardware and software, to give all the workstations the apparent power of the "richest" among them. For instance, if only one PC in a department has enough in main memory to run some program, users at any of the other PCs can (one at a time) access that program almost as if it were running on their own machines. In this case, the "rich" PC acts as the host machine, and the user's PC as the terminal.

The reason the coupler is effective in this situation is that it can "copy" the computing environment of the better-endowed workstation to any of the others. Resources which may be shared this way include

- Coprocessors
- Hard disks
- Large shared data bases
- Letter quality printers
- Plotters and other output devices
- Dial-up lines to various information services
- One-machine-only software

A slight modification to the setup protocol of the coupler is necessary to accomplish this sharing. In particular, the host workstation must be able to wait for a telephone call, begin waiting for an initiation request when a call comes in, and revert to the waiting state when the termination grant is received. These changes are relatively minor.

Some further considerations

Badly behaved applications. The prototype programs as described so far do not do the whole job. First of all, many commonly used programs on the PC workstations are not well-behaved, in that they do not use the standard interfaces for all their I/O operations. It is common for editors, for instance, to write directly to the display buffers (the PC displays are memory-mapped) rather than going through DOS or the BIOS. This method is necessary because of speed considerations (the BIOS interface has a one-character bandwidth) but complicates the job of the interception code. In one version of the prototype, this problem was solved by having the program in the host periodically examine the display buffers directly and inform the terminal of any changes found. A more serious problem occurs with some programs that directly access the I/O ports of the workstation to get input from the keyboard. There is no way to pass input that is received along the communications line to these programs in such a way that it will be "mistaken" for local input.

Speed. The speed of the communications link is another limiting factor in this solution. The current prototype performs tolerably when the workstations involved are physically close enough to be connected by a high-speed (4800 or 9600 bits per second) data link. In a typical remote application of the facility, though, the comparatively slow data rate available on telephone lines (1200 or fewer bits per second) restricts its usefulness. Implementing some kind of compression scheme on the line would help overcome this problem but would most likely worsen another: the amount of memory of the workstation occupied by the interception programs. Every byte used by these programs is a byte not available for applications. As larger memories in PC-based workstations become common, this problem will become less critical.

Other I/O devices. The display screen and the keyboard were chosen as the primary devices that are of concern to the user. Alternate input devices (mice, joysticks, panels) and sophisticated output devices (high-resolution graphics, speech synthesis) are becoming more common on business workstations, and a useful implementation of this solution will eventually have to deal with them. Finding the proper narrow paths for interception may be more difficult in these cases. Since the popular workstation operating systems were designed primarily for keyboard-and-screen environments, applications that use more advanced devices tend to communicate with them directly, for want of operating system support. The speaker on the IBM PC, for instance, is used by many programs, but it is not supported through either DOS or BIOS interfaces. The prototype coupler can therefore not support it. If a program is playing the Washington Post March in the host PC, users at the terminal PC will not hear it playing.

Further directions. Speed and reliability always offer room for improvement. The current prototype includes a checksum with every message interchanged, but the checksum is not put to much use. Error detection and correction, or a more sophisticated acknowledgment protocol, would increase the reliability of the link. It might also be desirable to have an automatic resynchronization protocol, to re-establish the integrity of the shared environment if either workstation detects a synchronization problem. Any of these additions would probably decrease the effective speed of the link.

Data compaction could be used to increase the effective speed, at the cost of more code space dedicated to the coupler in the host workstation (code space in the terminal workstation is effectively free).

A means of providing for direct communication between the workstation users was discussed earlier. It could be implemented in the coupler itself, at a small cost in reliability and transparency, or in an external medium.

Various modifications to implement the resource sharing use of the coupler, described previously, have also been tested. In theory, the control flow in the host program changes a bit; after receiving an "acknowledge shutdown" message, the host should go back to waiting for an initiation request. In practice, the modifications turn out to be a little more complex, involving the specific ability of various modems to automatically answer a telephone, break the connection, reinitialize internal registers reflecting the state of the communications link, and so on.

Several users of the prototype requested a means of actually transferring data from the storage devices of the host workstation to those of the terminal. This operation would involve establishing several new messages ("request for data transfer," "begin transfer," "transfer block," etc.), more careful checksum handling to ensure data integrity, and some kind of user interface to request the transfer. This represents a natural extension of the idea of duplicating the computing environment: the remote participants may access not only the I/O devices of the host workstation, but the storage devices (diskettes and fixed disk drives in these workstations) as well.

Conclusion

The use of tightly coupled workstations as presented here seems to solve a variety of problems associated with the distribution of computing power. As these problems, and therefore solutions of this type, become more widespread, it is hoped that operating systems will become more cooperative, in the sense of providing narrow paths to relevant user I/O devices, and enforcing the use of those paths by applications. (This is not, of course, the only reason to hope for clean interfaces.)

The problems addressed here are parts of a more general problem, which will require a variety of solutions. As computing power moves out of large central facilities and into workstations, access to the computing environment and to the information in those workstations becomes more difficult, hence

the profusion of local area networks (or at least designs for them) and communications programs. The solution outlined here applies to a very tightly coupled sharing of information and the computing environment; looser couplings will require solutions of their own.

Cited references and note

1. "Computing environment," as used here, includes the operating system, application programs, and relevant I/O devices such as keyboards and CRT screens.
2. *IBM Cooperative Viewing Facility, General Information*, GC34-2149 (Program Number 5664-187), IBM Corporation; available through IBM branch offices.
3. *IBM Personal Computer Technical Reference Manual*, 6024-005; *IBM Personal Computer/XT Technical Reference Manual*, 6936-808, IBM Corporation; available through IBM branch offices.
4. *IBM Personal Computer Disk Operating System, Version 2.1*, 6024-125, IBM Corporation; available through IBM branch offices.

Reprint Order No. G321-5223.

David M. Chess *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Chess joined IBM in July 1981 at the Research Center. Working at first on VM performance and workload management, he received an Outstanding Technical Achievement Award in 1982 for his contribution to the VM/370 Resource Limiter. When the IBM Personal Computer was announced, he became the Center's first Personal Computer consultant. He then joined the Advanced Workstation Projects group and is now manager of Advanced Workstation Services, working on new ways of increasing productivity through distributed workstations. Mr. Chess also maintains the "IBMPC" data base, a conferencing facility devoted to the IBM Personal Computer on the internal computer network, VNET.