An Introduction to ENVIRON/1

Cincom Systems, Inc./2181 Victory Parkway/Cincinnati, Ohio 45206/(513) 961-4110

SYSTEM INTRODUCTION

Business has long recognized the need for computer systems that are more responsive to their information processing requirements. Recent articles in many industry and business publications have bemoaned the fact that management's requests for pertinent information in useful formats is often not met by today's computer installations. A manager who wants information on which to base a decision often has to wait days for it. When he finally does get something, it is often meaningless, or at best an historical recap of past situations. If the delay has been great enough, the original requirements have changed and the cycle starts all over again. This type of reporting is hopelessly inadequate for the utilization of the computer as a management and control tool.

In an effort to obtain information on a more timely basis, many companies have implemented on-line, real-time, or terminal-oriented systems. These systems have the company's records and data readily available on a current basis for either examination or updating, and provide terminals at remote locations for people who require access to the information base. Examples of activities that are improved by this type of ability include order entry, customer status, production control, and airline or hotel/motel reservations.

Even though the objective of these systems is to provide information on a more timely basis, the results are not as satisfying as one might expect. Why? The processing is certainly not that difficult. The programming necessary to update an inventory record or to reserve a seat on an airplane or a room in a hotel is straight-forward when compared to the intricacies involved in even such old standing computer operations as calculating the payroll. Basically real-time application program logic is no more complex than traditional batch programs.

Why then are there so many more failures than successes when it comes to providing management with the kind of information it needs on a timely basis? Why aren't there more successful on-line, real-time, terminal-oriented systems? The answer is simply that the problems are outside the scope of the application processing of the company's information. They are obscure, hidden under the surface, and are the result of the new environment unique to on-line terminal-oriented processing. (Figure 1)

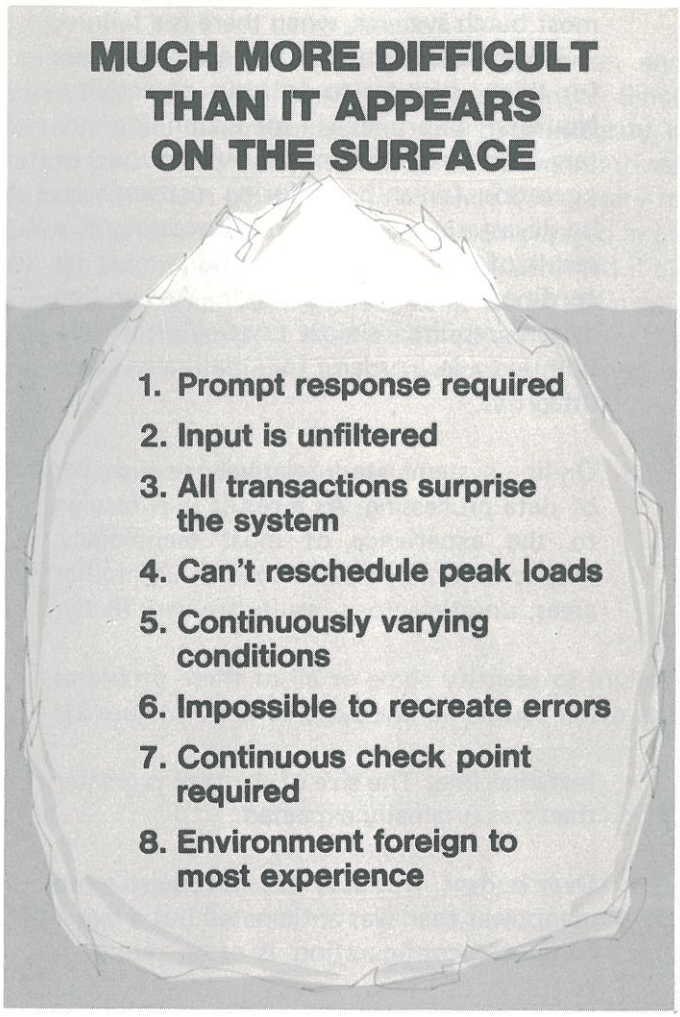

FIGURE 1

These problems include:

1. A prompt response is required to service on-line requests. The system must be prepared to interrupt other processing to respond to the on-line request. It must provide response time fast. Slow response is usually not acceptable. Mere seconds are critical.

2. The input to the computer from a remote terminal is unfiltered and unedited. In other types of processing, input is normally verified and corrected before the computer is required to process it. In an on-line environment the mistakes and errors made at the terminal must be handled by the computer. This requirement alone can add

significant load to programming the application.

3. The transactions that are handled in an on-line mode are a surprise to the computer system. It did not ask for the transaction nor did it have advance warning that a certain type of transaction was going to be presented for processing. This is completely different from conventional batch processing, where we usually sort a batch of input data prior to applying it to the data base or running the job.

4. Typically (in batch processing systems) when volume is greater than usual, work is rescheduled until a later time or the computer production schedule simply runs over time. With an on-line system, it is impossible to reschedule the peak loads. As a result, the system design is much more exciting and leaves less room for error. On-line input is volatile and may never be recorded prior to entering the system.

5. The conditions surrounding the on-line system are continually changing. The type of transactions that the system is being surprised with during one time of day varies from another time of day. The system is expected to be in operation during extended periods of time yet it must contend with failure of terminals, lines, storage devices, etc., and still not fail the user.

6. It is often impossible to recreate certain error conditions. This is significant. It means that programming errors are much more difficult to eliminate than they were in batch systems. Moreover, it increases the requirement for an automatic and effective recovery from problems when they arise.

7. In all data processing the ability to recover from error conditions is essential. The primary difference between recovery in a batch mode and recovery in an on-line mode is the time period available. In most batch systems, when there is a failure, it is very acceptable to rerun or reprocess for thirty minutes to an hour of activity. No such luxury exists for an on-line system. To have an on-line system out of operation for an hour during recovery can be devastating. Reruns or reprocessing as a result of on-line error conditions must be confined to a few seconds. An on-line system requires almost continuous checkpointing in order to maintain data integrity.

8. On-line systems are a relatively new aspect of data processing. As a result it is foreign to the experience of most technicians. Whenever work is performed in unfamiliar areas, unsatisfactory results are very likely.

Failure to identify some or all of these problems has often caused on-line systems to be (Figure 2):

- **Installed late.** The size of the task is greater than was originally expected.

- **Over budget.** Not only does it require more manpower than was anticipated but a larger computer configuration is often required.

- **Below performance specifications.** Even

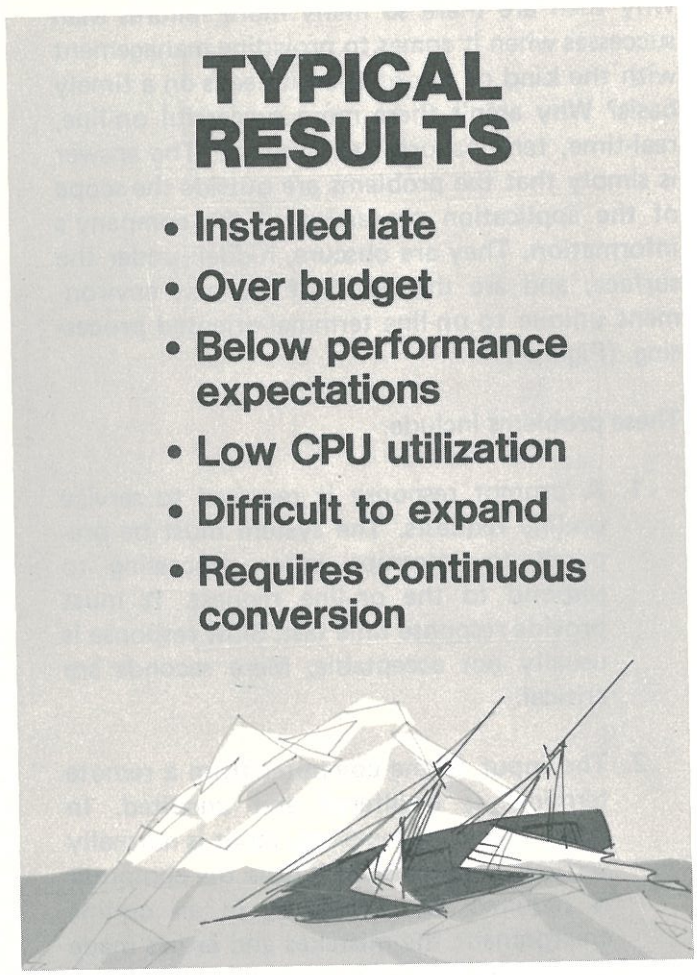

FIGURE 2

though a larger computer is necessary to do the required job, the response time that results is poorer than planned and fewer applications can be implemented than anticipated.

- **Low in actual computer utilization.** When a hardware monitor is used to measure the actual use of the CPU, the channels and the direct access devices, it is often found that utilization is under 20%. In spite of this, the system may be out of capacity.

- **Difficult to expand.** Additional applications all too often require modofication of previous applications. This occurs when on-line control functions become intertwined within the application programming.

- **Continuously under conversion.** Conversion to new terminals, conversion to new storage device, or new software approaches, conversion to new hardware or new operating system. This effort should and can be directed to developing new applications or solving new applications problems.

If it is possible to identify the problems that cause a system to be unsuccessful, it must also be possible to identify the requirements for successful real-time implementation. What are these requirements?

To be successful, a system must be **easy to implement and use.** If programmers are able to program in a familiar high level language, using techniques with which they have had experience, implementation time will be shortened. Advances in computers and peripheral equipment and operating systems are inevitable. It is important then, that the system maintain **maximum flexibility** to be able to take advantage of these without programming. Almost all on-line systems grow beyond their original objective. Many begin with only a few terminals and grow to encompass hundreds. A **broad performance range** is needed to allow for this growth. Conversion to handle growth must be eliminated.

As the on-line system becomes more and more an integral part of the company's operation and decision-making process, **high system availability** is a must. Fast recovery of system failure is essential. Since the company's data base may well be its most important asset, **maximum system and data integrity** is essential to assure its accuracy and protect the information from being viewed or modified by unauthorized personnel. The **ability to identify future requirements and the effect of changes in advance** is necessary to allow proper business decisions regarding future hardware changes. A continual battle is being waged in an effort to provide **optimum performance and effieiency.** Optimum performance can easily be achieved if all programs and data can be stored in the main storage of a computer, however, that is not efficient. Reasonable trade-offs are necessary. **Complete information system support** must be an objective from the outset and planned for in a series of evolutionary steps that lead ultimately to the realization of a system that meets management's needs.

Therefore, minimum requirements for a successful real-time implementation include:

- Easy to implement and use
- Maximum flexibility
- Broad performance range
- High system availability
- Maximum system and data integrity
- Ability to identify future requirements and effect of changes
- Optimum performance and efficiency
- Complete information system support

During the past several years many approaches have been followed in an attempt to provide a system that meets the above requirements.

Probably the first of these approaches is the single thread or one real-time program-at-a-time system. The single thread system reads a message from a remote station and processes that message to completion before going to the next message. A time graph representation of this kind of system is shown in Figure 3. Note the CPU time used compared to I/O time.

The computer initiates a poll of the communication lines attached and waits for a response. When a signal is received that the terminal is ready to transmit a message, the computer causes the terminal read to begin and waits until its completion. Following some superficial editing of the message, a program can be selected which will process the message. The computer must now wait until the program has been read into core and is ready for execution. With the program ready for exectuion, the actual processing of the message can begin. This will probably require that one or more records in the data base be read and possibly update. In any case, the actual CPU processing time is very small as compared to the I/O wait time. Following completion of processing, it is necessary to make a log or an aduit trail of what has taken place and write an answer to the terminal to inform the operator of the completion of processing.

Figure 3 shows an example where the overall response time is three seconds. During this time two records are read and updated and a log written. Assuming a moderate amount of processing by the application program at 360/Model 30 CPU speed, the total processing time used by the application and operating system is about 300 milliseconds or .3 seconds.

Simple arithmetic shows that the system has the capacity to process one message every three seconds. If the demands on the system should go beyond that, this approach is inadequate. With an application in which a terminal operator can enter a message every 30 seconds, with 8 of these terminals the computer system is occupied for 24 (8 terminals x 3 sec. to process) out of every 30 seconds. If the volume of work to be done requires 11 terminals, there is insufficient capacity to be successful. This is true in spite of the fact that there is very low utilization of the CPU itself. Even when there are messages waiting to be processed, the CPU is busy only 10% of the time.

By comparing this type of system to the list of minimum requirements, it is unsatisfactory. Obviously it does not allow for a broad performance range. It has the capability to handle a few terminals and does not

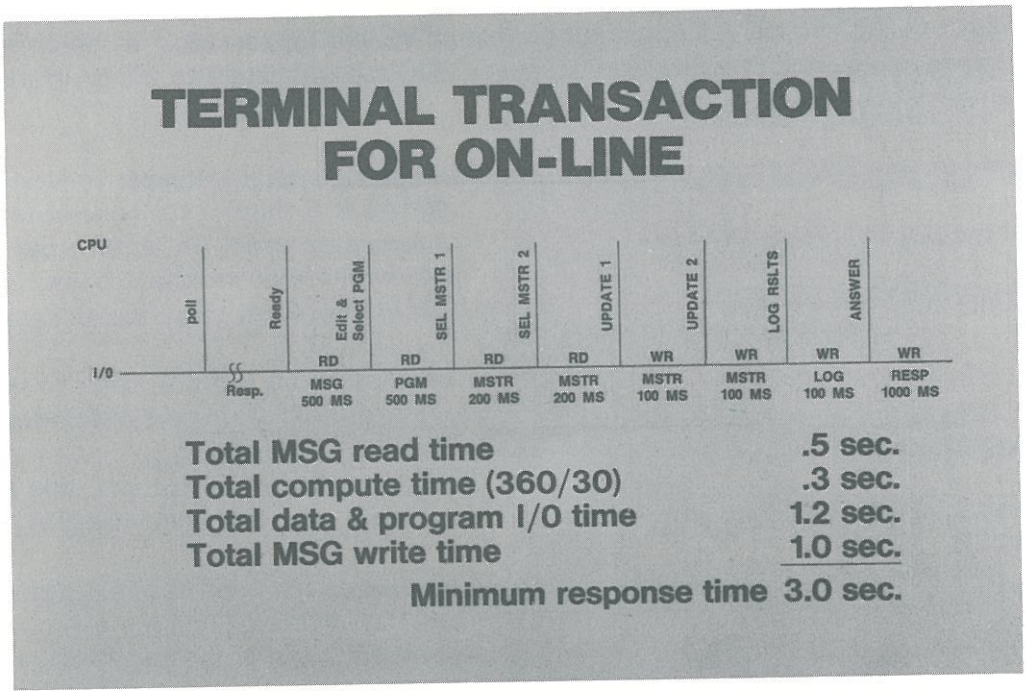

FIGURE 3

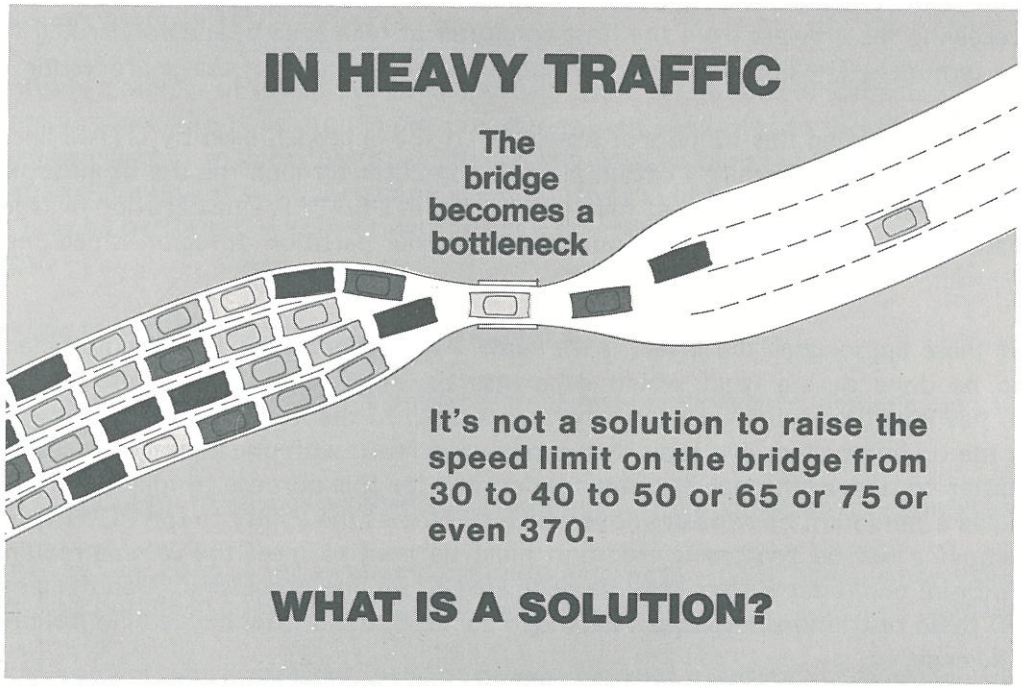

FIGURE 4

allow for expansion as the system grows. The system does not approach a balance between optimum performance and efficiency. A CPU that is utilized less than 10% and yet is out of capacity, is grossly inefficient.

Another typical characteristic of the single thread system is that while it is relatively easy to implement and use, it usually requires knowledge of communications lines, terminal devices, and specialized programming support. Usually the support of terminals and direct access storage devices is so intertwined with the application, any change in hardware configuration causes drastic program modifications. This limits the flexibility, provisions for high availability, data integrity, and identification of future requirements are often overlooked.

How can the performance of the real-time system be improved? Look at the same problem from another point of view. A similar situation exists where a four lane highway narrows to cross a one lane bridge. When traffic is light, automobiles cross the bridge with very little delay. However, during heavy traffic hours, the bridge becomes a bottleneck and the cars begin to stack up waiting to cross the bridge. It takes about the same amount of time to cross the bridge in heavy traffic as in light traffic, but the time waiting to cross the bridge becomes very long. (Figure 4)

How can the delays waiting to cross the bridge be eliminated? Very simply. Add additional lanes to the bridge. It is very easy to identify the solution to a low-performance single thread system. Add additional lanes for processing.

The Next Logical Approach – Functional Separation

It was shown in Figure 3 that approximately half of the three seconds required per message is spent in polling, reading, and answering the terminal. This is message handling. The other half was spent in processing and waiting for records to be read and written. A functional separation of the message handling and message processing functions can provide the equivalent of a two lane bridge. By separating these two functions and allowing them to be performed independently of one another, the capacity may be even doubled.

There are a number of ways to provide for the separation of message handling and message processing. One such approach is to provide a front-end computer in addition to the main or host computer. The front-end

computer's function is polling the terminals, reading the messages, passing them to the host computer at high speeds, receiving the answers from the host computer at high speeds, and forwarding the answers to the appropriate terminals. The host computer is responsible for the actual message processing.

Another method of providing this functional separation is the approach used by QTAM and TCAM. With this method, the separation is obtained within a single computer through the use of multiple partitions in DOS and OS/MFT and through the use of multiple regions in OS/MVT. One partition or region is used for the **message handling** and it operates independently of another partition or region which does the **message processing.**

With either of these approaches the effect is the same. Two activities can occur simultaneously allowing some work to be done during what would otherwise be wait time. As was described earlier, even a 360/Model 30 has much available time for other processing. In the event that the volume of work to be performed by the on-line system is greater than can be completed with one message processing partition, it is a simple matter to add additional partitions or regions for this purpose. Under DOS, the QTAM type approach requires a minimum of two partitions which eliminates the ability to run POWER or some similar spooling system. If a second processing partition must be used to meet the volume requirements, then, under DOS an entire computer system may be dedicated to on-line processing. With the greater capacities of OS/360/370 these restrictions have been extended some, but still offer a severe limitation to continued growth and efficiency.

In systems where the volume of activity against one application is very high, it is necessary to repeat copies of the programs for that application several times to provide the required number of lanes. This is not only wasteful of core space, but it also makes data security and data integrity almost impossible. The multiple partitions or regions execute independently of one another. Some technique must be devised to assure that the update from one partition will not be overlooked by the update from another. Facilities in the operating system do not provide a method for checkpointing and recovering data, when needed, because of either hardware or programming problems. Each of the partitions and regions are logically able to checkpoint and recover files that it owns and does not share with another partition. If multiple regions update the same file, complete automatic recovery cannot take place since each region can make updates without the knowledge of other regions. Obviously, the common data base concept where many users share common data is not feasible under this mode.

Implementation of this kind of system requires increased knowledge of specialized programming systems like QTAM, but the programming of the applications themselves need be no more difficult than a single thread system. Flexibility is increased over the single thread system since the terminal handling is automatically separated from the application programming. The performance range is definitely broader than a single thread system, although the overhead is often much higher. The performance limitations are caused by two factors: first, the number of partitions available; and second, the amount of core available.

System availability is not enhanced by this kind of system. Rapid automatic recovery from error is not possible if multiple partitions or regions are being used and the added complexity of the system contributes to the probability of errors occuring. System and data integrity are open to question since there is no central control available to assure their presence.

Nothing inherent in the system provides for identification of future requirements. The lack of central control does make this implementation difficult. The tradeoffs between optimum performance and efficiency is minimal. Code may be duplicated to provide multiple taks (lanes across the bridge). Core and number of partitions are valuable resources which are quickly spent.

This approach tends to be tailored to a specific installation's requirements. Except for the problems stated above, this would seem to be an advantage. It may, in fact, be another disadvantage since the approach may not be compatible with generalized solutions to other problems or new application requirements. If not, major conversions will retard further new application development. All of the problems we see here point out a need for a supervisor or monitor.

The Partition Supervisor

The important shortcomings of the QTAM-type approach were: 1) **no central control,** 2) **poor utilization of partitions and core space.** The partition supervisor can present a solution to these by controlling the entire real-time operation within a single partition and adding core as required. The functional separation of QTAM-type approach remains an important concept, but instead of being performed in its own partition or region, it can now be performed within a single one as a function of a partition supervisor, control program, or task monitor.

The major functions of the control program supervisor are:

1. Telecommunication control of information transmission through communication lines to remote terminals.

2. Message distribution to various programs such as inquiry, order entry, or file maintenance.

3. Computer facility control for allocating computer resources — CPU, core memory, channels, and I/O devices. This is done in a fashion permitting overlapped processing of a number of messages.

4. Data management which provides data for processing from a variety of file organization techniques.

There are a number of techniques that are used in the design of control programs or partition supervisors. The method used by most airline reservation systems is perhaps the best. Using this technique a message is read and is processed until I/O is required before further processing can continue. At this point the I/O is initiated by the control program and an examination is made to see if other messages are either ready to begin processing or to continue processing, as a result of I/O requests being satisfied. Most airline reservation systems use a version of PARS (Programmed Airline Reservation System). This is a high performance system requiring a specialized operating system and special hardware features.

This type of system utilizes reusable code to prevent duplicating programs or core. The partition supervisor acts as a policeman and directs the flow of traffic. When the first message is being processed, the program is pointed by registers to that message and the data associated with it. While I/O operations are being performed to satisfy processing of the first message, the control program may elect to use that same program to process a second message. This is accomplished by pointing the registers to the second message and its associated data. Programs written to operate in this environment are said to be re-entrantable. Instructions are not modified and only data areas that are defined externally to the code can be changed. This external definition is via a linkage section in COBOL or via a DSECT in assembler language.

The introduction of a partition supervisor or task management system opens up a method of solving many of the problems inherent in transaction driven systems. Data and system integrity can be closely controlled. System restart and continuous transaction logging can be accomplished. Even though most systems available today do not provide a satisfactory solution to this problem.

There are some problems that are introduced and magnified by the partition supervisor which were not apparent in previous systems we discussed. One of these is the difficulty of using high-level languages such as COBOL. The PARS system, for example, uses one of the most advanced of all real-time control programs, but the performance, structure, and function that was required of the application programs eliminated the ability to use standard compilers. This difficulty is dramatized by the fact that almost all systems of this type have the application programs written in an Assembler language. Other applications found it necessary to develop their own paged, re-entrant version of a high-level language in order to realistically maintain their real-time system.

A summary of the problems that are left unsolved or which are introduced by partition supervisors are listed below:

1. Partition fragmentation
2. Program loading overhead
3. Unused program code
4. No task context capability

Partition Fragmentation

Figure 5 shows a real-time partition or region in the midst of operation. It shows programs being used to process several messages. Task 1 and Task 4 are being serviced by the same program simultaneously. All of the programs are obviously not the same size. Also, all of the programs do not require the same amount of data to process a message. Figure 6 shows the same system following the completion of processing on two of the messages. The core that was occupied by Task 2 and 4 has become available for re-use; however, these fragments of core are still bounded by active core. They can be used only if the task waiting to run is small enough to fit in a free fragment. Even though many small areas of core have been freed up, our waiting task may have to wait for a larger contiguous block to become available. Anytime a system suffers from this condition, we find an extremely poor utilization of core. Fragmentation often gets so bad that system performance degrades to a point where an operator must bring the system down and manually restart one task at a time.

Partition fragmentation is a very complex problem to solve. Most real-time control programs suggest that the problem be avoided by making all programs the same size. This assures that when one program is no longer needed the next program to be loaded will fit in the same core space. The solution, making all programs the same size, may be worse than the problem. What size should the programs be? If all programs are made to be the largest size, then obviously much memory is wasted. If all programs are not made the maximum size, then programming effort is wasted in attempting to squeeze the programs into some smaller size than would normally be required. This adds significantly to programming costs and development time.

Another method sometimes used to avoid fragmentation is to have the user organize his programs into an overlay structure. This allows various program sizes, but forces the user to identify the programs that cannot exist simultaneously in memory. Performance is impacted whenever the terminal demand requires programs which have been identified as mutually exclusive. The problem must be solved to prevent complicating the application programming and degrading system performance through poor core utilization.

Program Loading Overhead

Dynamic loading is a major requirement in a real-time system. Messages arrive at the computer in an unpredictable sequence and at unpredictable times. The partition supervisor or real-time control program has the responsibility of assuring that the required programs are available to process the messages as they arrive. It is possible, of course, to have all programs that will ever be required to process the on-line actively resident in main memory. It is possible, but impractical, except in very small application. Since even small applications will grow, they should also be implemented using techniques which will eventually allow the programs used in the system to exceed the main memory space allowed at execution time.

As the number of applications in the on-line system increase, the amount of time that the system spends loading programs increases dramatically. It is not unusual for high activity on-line systems to spend 30-40% of the time loading programs. This, of course, affects performance and response time.

When we look at our applications programs we see that they are designed to handle all possible exceptions. There are many branches both legs of which will seldom be used to process the same transaction. Studies show that only 5-10% of the program is executed during the processing of a given transaction. This also

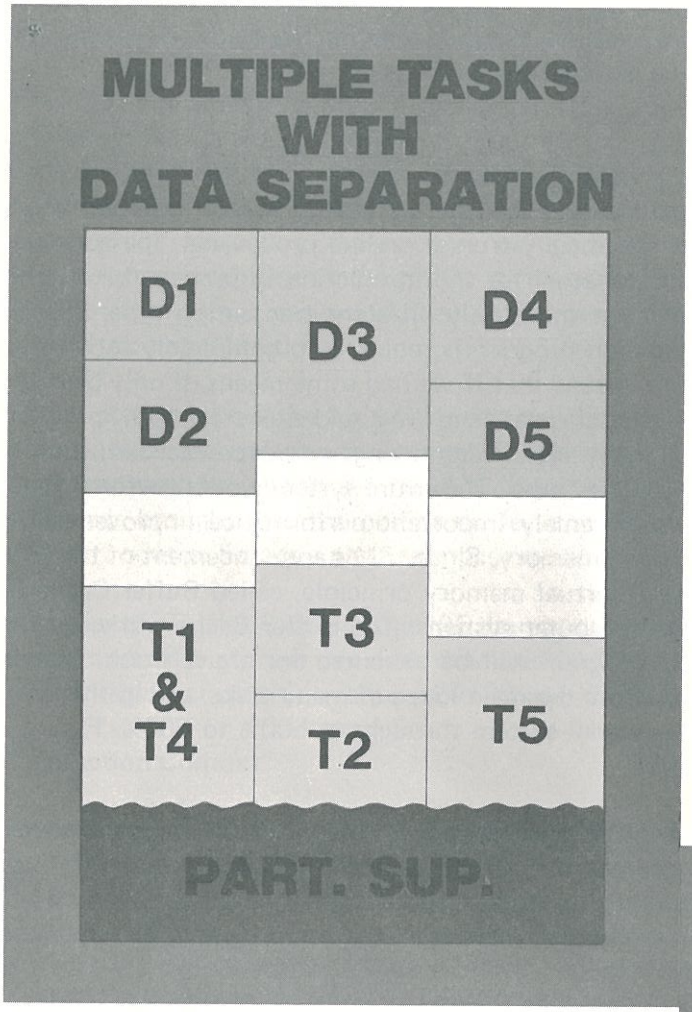

FIGURE 5

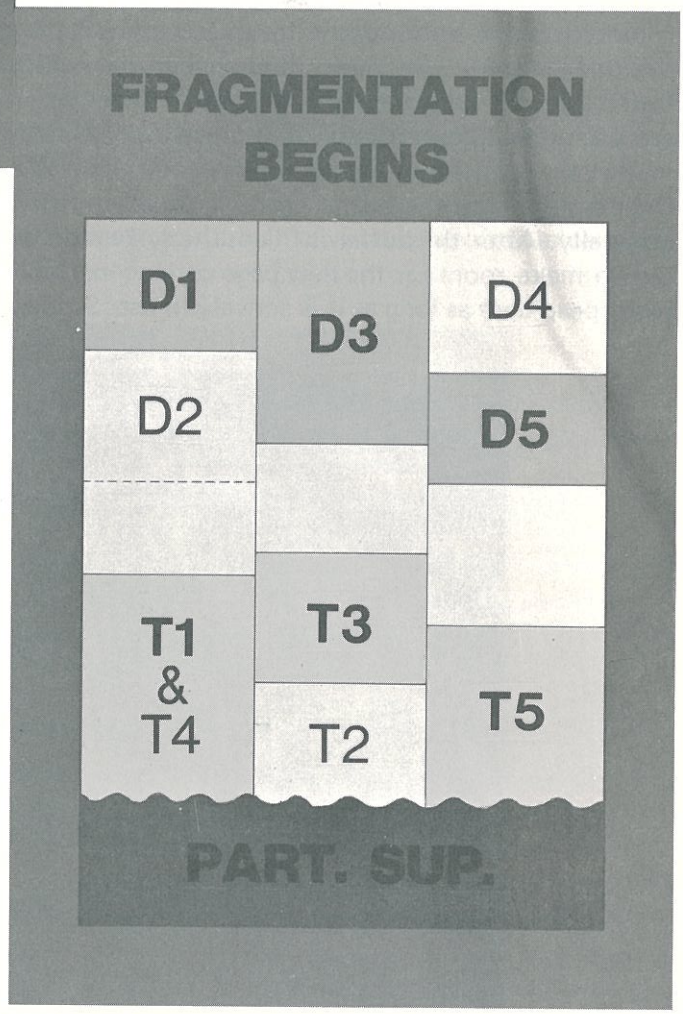

FIGURE 6

means that when we load the entire application program we load many times as much code as we expect to use to process the transaction at hand. We are not only wasting the load time required but one task is now occupying tremendous amounts of core that will just lie inactive during the entire process time.

Unused Program Code

As we have mentioned before, every program is written using branches and every branch represents two separate paths of code that could be taken. This is specifically true in on-line processing. The code not branched to its exception code that may not be executed at all for the transaction in process. Likely this code is there only to handle a specific exception or even an entirely different transaction type. Studies repeatedly show that less than 10% of a typical application program is required to completely satisfy the processing requirements for any given transaction. This means that if we had some means of only bringing in to core those instructions actually needed for the transaction at hand we could execute a 100K program in 10K or often less than 10K. With this capability many significant savings could be realized, such as greatly reduced core program load time, and faster response time. The entire system would perform much more efficiently. The technology to accomplish this extremely important performance improvement is available. It is often referred to as virtual memory or paged memory. Since IBM's announcement of the IBM 360/85 or 195, hardware implementations of a type of virtual memory principle, called Buffer Cache or Buffer Store have been employed in these very large computer systems. The Buffer Cache works on the principle that only a small part, 10% or less, of the program will be executed for a given task. On the 360/195 a 32K Buffer Cache provides enough space to store the main loops of many tasks, and in this way, a small high speed memory can actually increase the overall system throughput 500% to 600%. Figure 7 illustrates how the 360/195 and 370 Buffer Cache works.

A large main memory contains the programs and data initially. As executions begin, a small page (64 bytes) is moved to the buffer cache for execution. As the program branches from point to point, the small fixed sized pages containing the code branched to is moved into the buffer where it can be executed many times faster than in main memory. The main memory is essentially a storage area for programs and data and everything that is executed is first moved to the buffer cache.

Once a page is in the buffer cache it may be reused over and over each time saving the fetch time required originally. After the buffer is filled, the system determines the oldest page without use and throws this page out to make room for the new page coming in. This enables the main loop (series of pages) to remain in the high speed core as long as it is actively in use. Studies show that the addition of a relatively small high speed

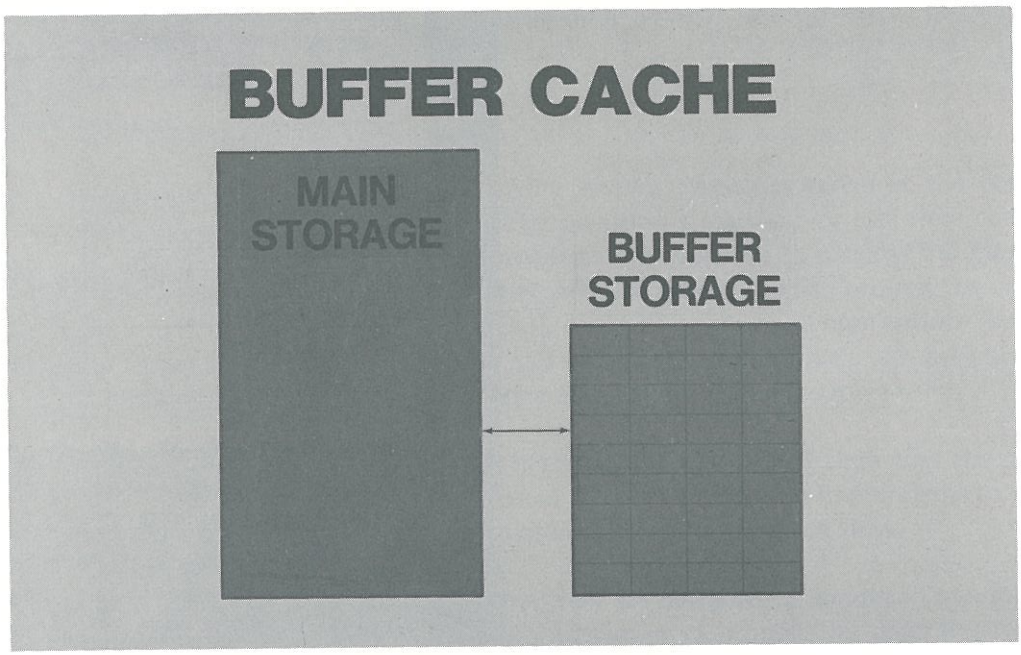

FIGURE 7

buffer cache to a main or large storage will produce execution speeds very close to the high speed buffer storage, allowing a small core area to provide the performance of one many times larger.

If we explore this principle a little further, (See Figure 8) we see that the same relationship exists between the direct access storage devices on your system and the main memory or real-time region — a large high capacity relatively slow storage device and a high speed relatively small core region. By loading small fixed sized pages of the programs stored on the direct access storage devices into core as they are needed and dynamically relocating these pages as space is made available we can achieve the extremely high performance capability of the buffer cache concept via software implementation. Also, by simply keeping track of the various pages and their usage we can allow reuse of the pages already in core and siginificantly reduce program load time.

Figure 9 shows how the buffer concept works with ENVIRON/1. All programs and data are stored on the direct access storage devices in small fixed sized pages. As these pages are needed, they are read into the real-time partition one at a time. Any pages not required for the current transaction are left out on disk. Any page brought into the on-line buffer store is available for reuse by the current task or any other task that gets control before a particular page required becomes the oldest page in core without use.

Just like the Buffer Cache in the 195 dramatically increased performance, the ENVIRON/1 real-time buffer cache or buffer storage provides dramatic performance increases and significantly reduces the amount of on-line region or partition required to achieve this high performance.

Transaction Context

There are many examples we could use here to explain the vital requirement for transaction context in any form of data processing. However, on-line or transaction driven processing seriously compounds the need. On-line processing is similar in situation to an executive trying to talk to a room full of people all at once with the condition that no one would say more than a few words before going into the next conversation. I am sure we can all visualize how close to impossible this situation would be for the executive to maintain context of even one let alone the many conversations. In real-time many different transactions may at any given time in various stages of progress go through one or more programs. The extra burden placed upon the application programming to maintain context in the transaction driven environment often exceeds the extent of the entire application program in a batch environment.

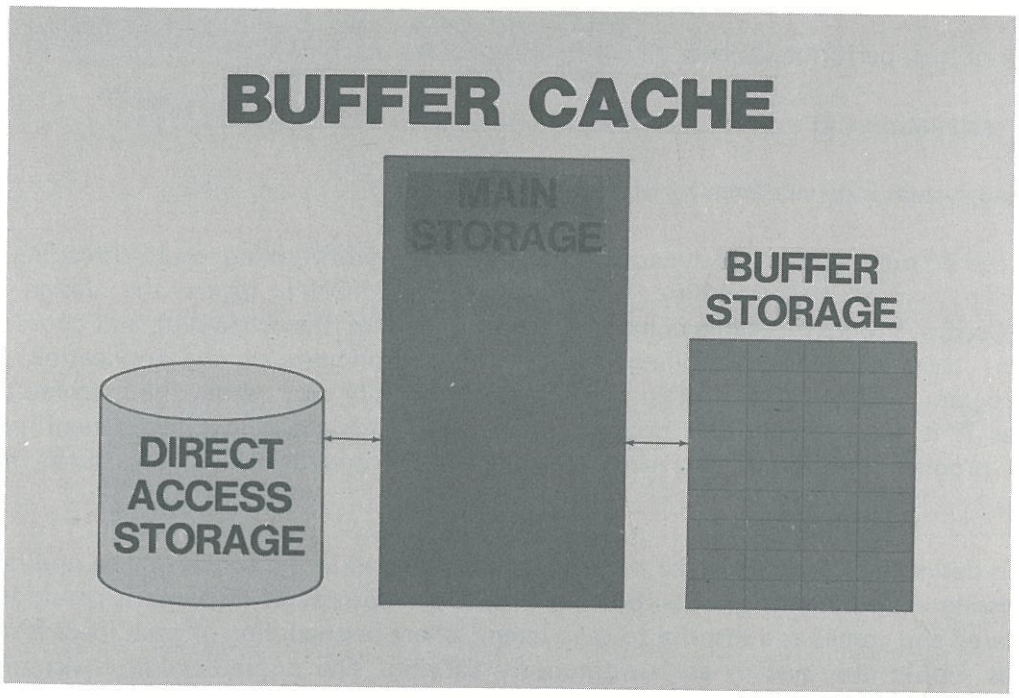

FIGURE 8

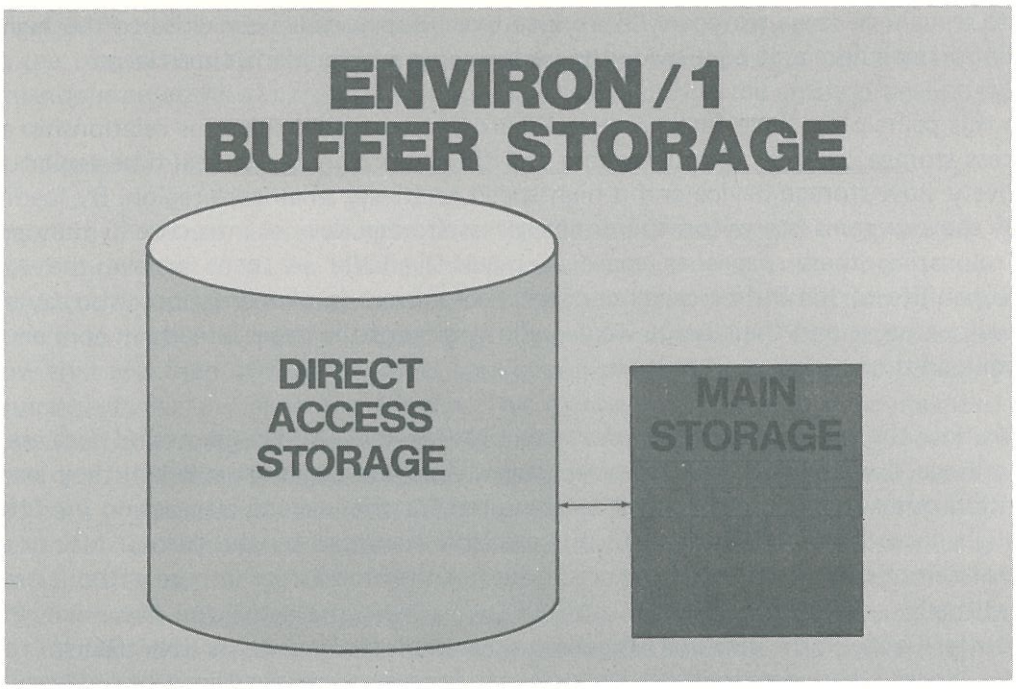

FIGURE 9

If our executive were to use his secretary to keep notes as he progressed through each conversation, he could quickly review what went on before and continue in context. Each time he was required to continue an interrupted conversation. This facility can be implemented as a function of the partition supervisor. Automatic context or a scratch pad for each terminal can be maintained for use each time a particular task gains control for execution.

ENVIRON/1 was developed to provide IBM 360 and 370 users with a means to have a real-time telecommunications system with the following design criteria:

- Efficient performance in a small region or partition
- Ability to add terminals and/or files with no reprogramming
- Delivery of high performance over all configurations and machine sizes
- Ease of installation and use
- Operating system independence for application purposes

Through the use of such features as dynamic core allocation, multi-tasking, multi-threading, buffer store, program sharing, virtual addressability, and paging, ENVIRON/1 meets the design criteria. The ENVIRON/1 Control Program accepts complete responsibility for the scheduling and coordination of the many different tasks and for controlling the general environment of the application program. The application program is allowed to function as if it were the only user within the machine and the entire system acts as if it were serving that program exclusively. This capability means significant savings of operator and programming time, plus simpler application programs which, of course, means simpler maintenance.

ENVIRON/1 is designed to ease the major differences encountered in the conversion to on-line system from a batch processing environment: an environment where promptness of response is paramount, where all input is unfiltered and comes as a surprise to the system, where rescheduling of peak loads is impossible and the conditions within the system are continuously varying. The unpredictable environment causes a problem for reconstruction of error conditions or debugging since most events are difficult if not impossible

to reproduce. This demands an almost continuous checkpoint in order to have any capability to recover from error conditions. Since these considerations are foreign to a batch processing system, the conversion to on-line processing can become a nightmare if experienced knowledge is not available. ENVIRON/1 provides the means to ease the conversion by controlling the environment to a large extent for the user. In the following sections, the concepts used within ENVIRON/1 will be described to illustrate the simplicity and the totality of this control.

Virtual System

ENVIRON/1 embodies the virtual system concept, which involves two viewpoints. First, the individual terminal operator has a one-to-one relationship with the computer as if there were only one terminal operator on the system. Second, the programmer writes application programs as if he were the only one using the machine and as though he had unlimited core storage available. The system acts as if it were serving only one terminal or station. The programmer sees the same one-to-one correspondence that the terminal operator sees, as if there were only one station to be concerned with, although in fact, the programs may be used by several stations concurrently.

A station is defined to be one or more terminals and/or devices and basically has only one user. The station-user's work, together with the programs which process that work, constitute an individual and independent portion of work identified as a task. Each task may be scheduled independently from any other processing that may be occuring within the system. A combination or series of small pieces of conversation to and from the terminal operator throughout the day constitutes one task's work. This conversation is divided into messages and responses. The minimum quantity of data that is sent from the terminal to the computer before a response can be received represents what is referred to as a message. The response is the data that is received by the terminal before the next entry at the terminal can be made.

It is context which makes the virtual system concept possible. **Context is essential; without context reliable virtual systems are impossible.** A conversation is meaningless if just one sentence is heard without knowledge of what is said previously. This status or 'context' of a conversation between the terminal and the computer must be maintained (i.e., each message must be linked with its unique context) between messages. If the context of a conversation is not maintained, the terminal operator must re-enter all information rather than just the latest piece of the conversation. Context is also required to maintain the status of each accountable terminal transaction as defined by the application program. An accountable terminal transaction is the basic element of accountable work within the system. For example, in an airline reservation system, the sale of a seat is the basic accountable transaction. A single transaction does not necessarily coincide with a message. It might be one message, many messages, or there may be many transactions within a message.

There is an accountable unit that is greater than a single transaction; many transactions together constitute a day's work. A day's work could consist of batch totals, sales per clerk or total amount of money accumulated, etc. These types of totals cannot be obtained without context. Context items consist of any status information that is established from previous messages or any associated data items previously placed in the context. The current position within the program and its associated status are also part of the context.

In a real-time, multi-terminal system, transaction messages are expected to arrive asynchronously from terminals attached to the computer. Since the physical input of messages from any given terminal requires a relatively long time (seconds) the high internal speed and low overhead of the ENVIRON/1 Control Program makes it possible to overlap the requests from several terminals, allowing each terminal to appear as if it were the only terminal in the system. Each individual station represents a task to the Control Program. ENVIRON/1 provides interleaved execution of several programming tasks apparently simultaneously (multi-tasking). Multi-tasking helps make it possible to utilize the channels and devices in the system to their full capacity.

System Flow

At system initialization time ENVIRON/1 sets up control blocks, checks for the presence of on-line data sets and initiates continuous polling of terminals. The first terminal read for each station is issued by ENVIRON/1. When the first input message from each station is received, the task for that station is

established, i.e., system context blocks are created. Each station then has its unique set of context containing its status, including current instruction counter and working storage. If a station never enters a message, context blocks are never constructed for it. Each of these initial messages is then sent to the user's Sign-on Routine, if it exists. This is a user-written routine designed to perform general housekeeping and to determine which routines will process this particular message.

A Sign-on Routine generally opens the on-line data sets required for processing this message, initializes counters and switches, makes necessary security checks, and may set up additional context blocks for this particular task. Each station must be routed through the Sign-on Routine to initialize its context, open its files, and possibly be assigned to an application. An application may represent a particular set of processing code, a particular manner of processing (i.e., read-only), or the processing against a particular file. The Sign-on Routine is designed to perform a set of one-time-only functions for each station. The station is then placed in a continuous conversation with its processing programs referred to as the mainline loop.

The mainline loop represents the normal editing and processing of this particular station's input message(s) where exceptions should cause branches to special routines. This mainline loop begins with the receipt of the input message (completion of the terminal read) and ends with output of the response to the station (completion of the terminal write). In this manner, a continuous conversation is maintained by requesting the next message at the same time that the reply for the current message is sent. Normally the next terminal read starts processing at the beginning of the mainline without ever returning to the Sign-on Routine again. This Terminal Read-Write loop has several advantages to the user: 1) the station does **not** need to keep requesting a particular program or reminding the system of its intentions, ENVIRON/1 remembers; 2) the amount of code which must be core-resident is thereby reduced; and 3) the program can be written as if there were only a single station and a single task, although in reality many stations may be executing within the same application program at one time. Initially, the Sign-on Routine will be the only program in core, but as each station begins its processing loop, the Sign-on Routine becomes inactive and is eventually overlaid by some other block. A program page becomes eligible for overlay when it is the program which has gone the longest time without usage. Since programs are re-entrant, there is no need for multiple program copies, thus saving core space.

Message processing occurs within the mainline loop. The nature of this processing is application dependent. The processing generally involves message editing and communication with a data base. The message edit facility of ENVIRON/1 is designed to provide a convenient means for editing each message. The processing of a particular transaction may also involve some working storage and additional context blocks. These blocks are obtained through the use of ENVIRON/1 commands. The attempt to simplify message processing to the issuance of a prescribed set of commands is intended to ease the programmer's job.

In order to end processing in a particular mainline loop, each station need only sign-off the particular application and sign-on to another. The sign-off procedure generally involves closing files, freeing up obtained blocks and perhaps storing counters in a file for later use. Since each station had individually opened its necessary files; it should, in turn, close these files, thus updating their descriptors if necessary. When all application housekeeping is completed, the final output message is written to the station without an accompanying terminal read and no return point. The station in question is now free of any application loop although other stations continue to process within the same mainline until they have completed their processing. The free station may now sign-on to another set of processing routines by entering the Sign-on Routine (which is where any subsequent messages will be directed automatically). When all stations have signed off an application, it is considered closed. Normally the intention to close an application is indicated by the system operator through a console message. The message is interpreted by a user-written routine which should cause a close application command to be executed to alert the system to begin application termination. Each mainline loop executes a test for application ending which in turn activates the sign-off routine for that mainline and allows the station to finish processing.

Concept of Application

These mainline loops generally represent specific applications within the system, perhaps dedicated to

certain terminals; i.e., a group of terminals may be performing data entry and another group an inquiry function. ENVIRON/1 centrally controls all these applications by associating an application number with each separately defined function. It is possible to open an application, quickly obtain desired application program(s), assign terminals to appropriate applications, inquire how many and which stations are involved with a particular application and close an application. The concept of an application allows the user to direct and control system functions with ease through a master terminal or application supervisors. The master terminal may dynamically monitor the operation of the system in inquiring how many and which stations are involved with each application, open and close applications, as well as turn on and off lines and terminals and dispatch other information to terminals. The facility to perform these operations is provided by ENVIRON/1; however, the implementation of the facility is left to the user since any application requirements are specific to a particular installation. Like the Sign-on and Sign-off procedures, these routines are user-written with guidelines and suggestions from ENVIRON/1 documentation. Figure 10 suggests a general structure for processing consisting of the Sign-on Routine including general housekeeping functions, specific application housekeeping routines and the mainloop of the application. Specific application housekeeping includes opening application files and possibly obtaining additional context blocks. The task is then placed in its terminal read-write loop. Upon application termination, the application sign-off procedure is activated. The test for application termination is performed at the beginning and end of the mainline loop in order to catch any terminal just signing on for the first time as well as each terminal as it finishes its last message. An operator's communication routing is also suggested for interpreting master terminal messages to application operations such as closing an application or terminal status inquiries.

Rollin/Rollout

Between the execution of each mainline loop, there is a time lapse when the response has been received and before the next entry is made. This time may be a matter of seconds, minutes, or hours; meanwhile, the context blocks for that particular station are unused. Since they are not currently in use there is no reason to tie up core with them. For this reason, a context or long-wait roll-out normally occurs following the terminal read request. These context blocks are placed in a Roll File awaiting the next action of their associated station. By placing this status information in a file, a checkpoint is obtained automatically between transactions in case of a later failure.

The Roll File also contains blocks which have been rolled out during processing, that is forced roll-out where in order for one task to continue processing, more space must be made available for it in core. Since a task probably has accountability or owns data blocks or extra working blocks during processing, the forced roll-out involves more blocks than the context roll-out. Hopefully, a forced roll-out condition is an uncommon occurence in most systems. If it does become common, it may indicate the need for more space in the on-line partition. The Roll File also contains space for program check dumps, if desired. During testing it is useful to obtain a record of application program checks, their probable cause and a snapshot of the core blocks owned by the terminal at failure time. The ABEND portion of the Roll File has a wraparound structure so that only a certain number of dumps may be taken (dependent on space allocated) and then previous dumps will be overlaid.

Program and Data Paging

The Roll File is only one of two necessary files for on-line execution. The On-Line Application Program Data Set must also be present for rapid retrieval of program pages since all programs are **not** core-resident. As a program page is needed, it is either branched to in core or loaded from the Program Data Set. The mix of programs in core is then determined by the application processing at that moment. As soon as core is full, the page that has gone the longest time without use, probably an initialization page, would be thrown out to make space for the next page requested and so forth. This tends to bring the mainline loops into core and as long as they are used afterwards, they stay in core. Every time a page already in core is reused, the time required to fetch it from disk is saved, and in high-activity programs this percentage of buffer store hits, as we call it, is very high. The same algorithm applies to data as it can also be reused without having to access it again, even by separate tasks. If a data record has been updated by a previous task after writeback

SIGN-ON AND MAINLINE PROCESSING LOOP

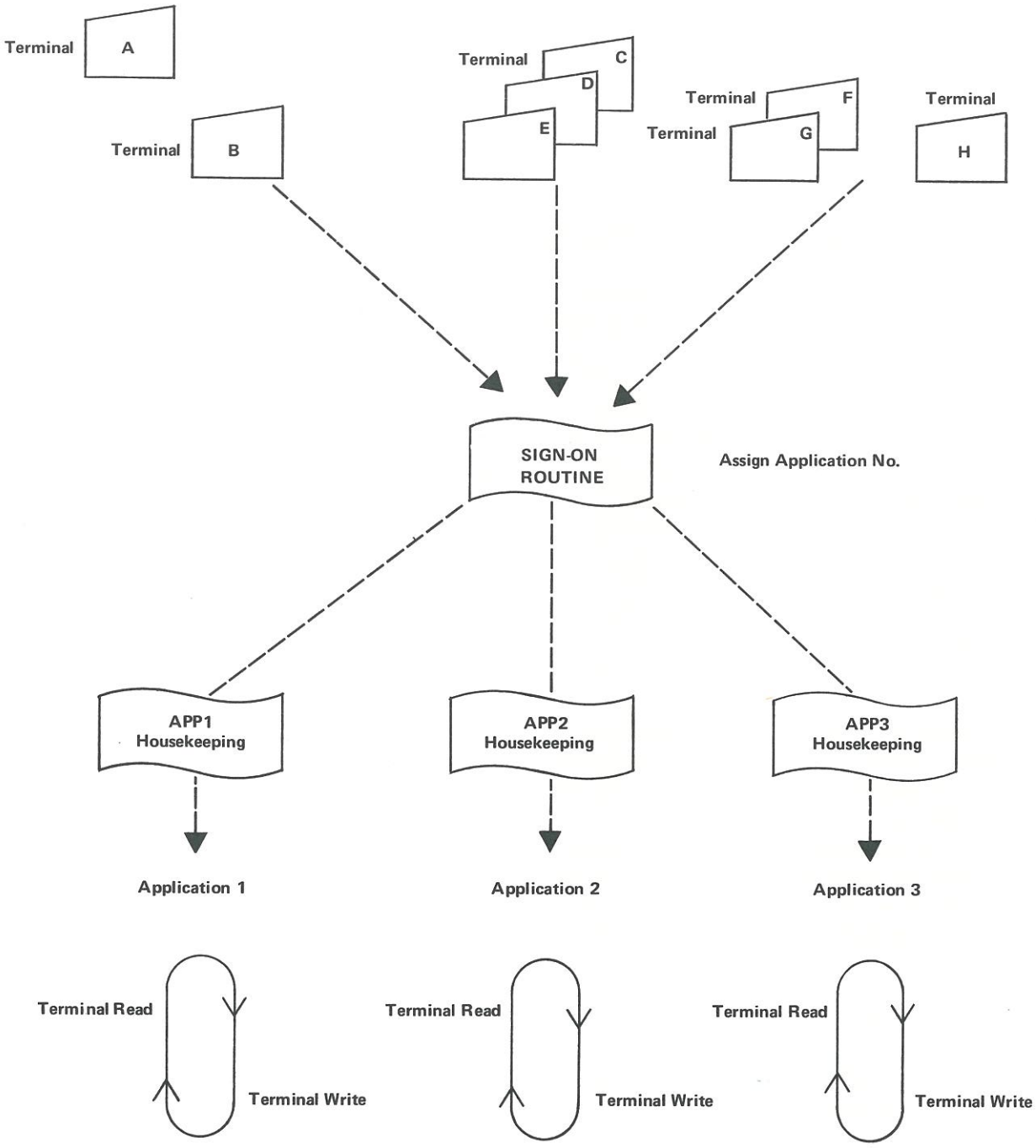

FIGURE 10

to disk, it still remains in core for potential reuse. Since programs are re-entrant, there is no need for program roll-out. A program page (or a data block) may have multiple users at any one time, so that any station's work may coincide with any other station's work without logical conflict for core space, thus allowing maximum core utilization. Another one of the purposes of context is to keep track of which station is executing which program page and accessing which data blocks at any point in time.

On-Line Data Sets

Generally, some on-line data sets are also present within the system for execution. These data sets may be created in off-line processing and read or updated on-line, or they may be created on-line for later batch processing. There are basically three types within ENVIRON/1, each of which may be accessed through at least two different access methods. The data sets types are TOTAL (network-structured); indexed (hierarchical structured); and serial (sequentially structured), either private or community. Each of these data sets may be accessed physically by use of a virtual address (physical position within the file) or logically by use of an index or another form of pointer.

The on-line data sets are described to the system in three ways: first, through the ENVIRON/1 control cards at execution time; secondly, through the data set descriptors present within each file; and thirdly, through the operating system job control language. The system opens these files at system initialization ensuring that they are present and complete. A logical connection between each task and the data set is established when the related application is initiated. In this way a file or files are easily removed from a particular day's operation by minor changes to one control card if the application is not to be run.

Whenever two or more tasks are processed simultaneously against a shared data set, there is a possibility for conflict of data. Conflicts of this nature must be controlled to ensure the logical integrity of the shared data base. ENVIRON/1 solves this problem with "data buffer store." This feature not only resolves the conflicts for data, but ensures that two stations requiring the same data have it available when it is needed. Shared data bases present no present problem when data is only referenced by different stations. Conflicts arise when shared data bases may be updated by one or more stations. ENVIRON/1 allows first one station then the other to have access to the data on a priority basis. The data buffer store feature also capitalizes on any inactive core space in order to retain data which may be used again by either the same or a different task, thus eliminating the necessity for reaccessing the data for every request.

ENVIRON/1 combines the allocation of working storage with the execution of each request for service from the system. This combining operation allows the control program to delay allocation to a more convenient time during the processing of the request. The core space involved does not sit idle during the operation. For example, in reading a record from disk, there is no need to have a block assigned to the read request until the direct access device has been allocated to this request and the seek has been completed. The waiting time for the seek and the device is far longer than the actual reading of the data. By not allocating the block at the beginning of the request, ENVIRON/1 enables that area to be used by some other station, thus cutting down the average core occupancy per station. This facility is known as "late binding."

ENVIRON/1 also allows multiple I/O requests to be made at once. So, if a program needs two or three data blocks to examine or modify at once it yields control until all requested data blocks are in core. Multi-requesting aids in efficient processing by permitting seeks to separate drives to occur in parallel rather than serially. This technique decreases potential wait time for all stations, thus improving system throughput.

The application portion of the ENVIRON/1 partition is divided into fixed length pages as shown in Figure 11. These core blocks may hold a program page, a data block, or context information interchangeably (all information being relocatable). A particular terminal may own context blocks and input or output messages but shares data block(s) and program pages. Since each type of page is the same size and may be occupied by data, context, or programs, maximum flexibility is provided as well as the elimination of core fragmentation.

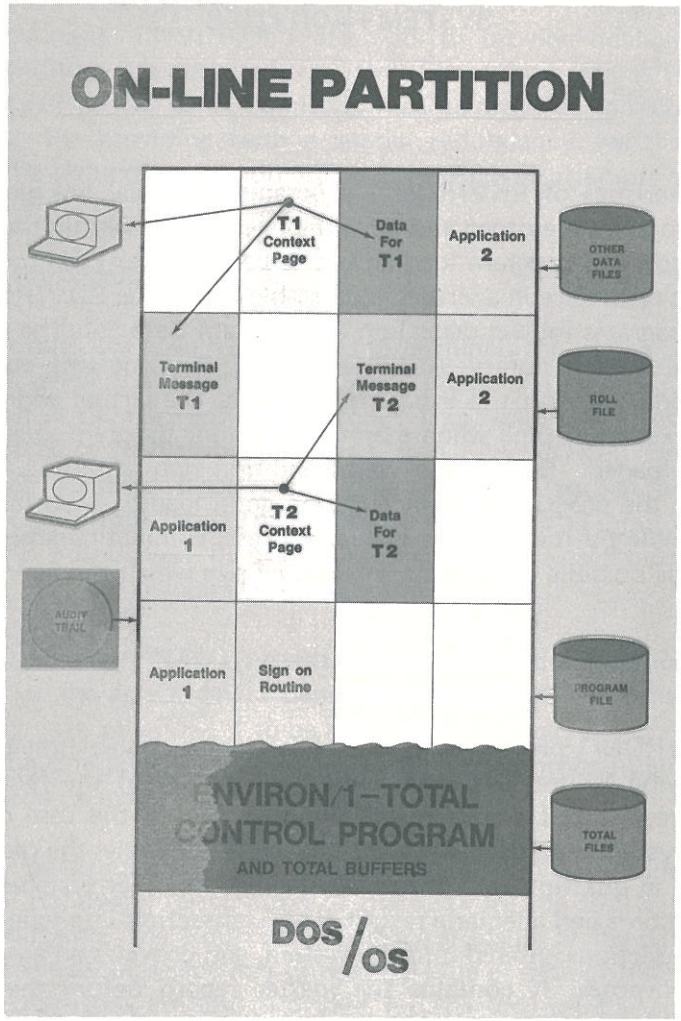
ON-LINE PARTITION
T1 Context Page
Data For T1
Application 2
OTHER DATA FILES
Terminal Message T1
Terminal Message T2
Application 2
ROLL FILE
Application 1
T2 Context Page
Data For T2
AUDIT TRAIL
Application 1
Sign on Routine
PROGRAM FILE
ENVIRON/1–TOTAL CONTROL PROGRAM
AND TOTAL BUFFERS
TOTAL FILES
DOS/OS

FIGURE 11

Linkage Editor

Due to the unique characteristics of ENVIRON/1 programs, a special linkage editor is provided. Since ENVIRON/1 application programs are paged and relocatable anywhere within the on-line partition, a means for the fast address resolution and program loading has to be available. The Program Data Set contains the entire on-line program page repertoire in a virtual addressable form. The ENVIRON/1 linkage editor is used to place these assembled programs (object decks) on the Program Data Set. The programs may exist in card or card image form on a relocatable library. At link-edit time, the programs, entry points, and all external symbols are given virtual addresses on the Program Data Set. The virtual address is not resolved into an actual core location until execution time when a branch from one program to another program not in core causes a new page to be loaded. There is no wait condition between these programs, i.e., no register restoration is necessary by the programmer, since the program loading is so fast. The rapid retrieval of a program page is made possible by the use of a virtual address, eliminating any table searching. The rapid loading of a program page is possible due to the small fixed size which can be loaded quickly and into any block available in core.

Restart/Error Recovery

The restart option of ENVIRON/1 provides the user with a quick, reliable means of restarting the system and recovering data following failure. Depending upon the nature of the failure, (system, partition, or application program) the restart and recovery techniques differ. In the case of an application program failure in an inquiry only system, i.e., no file update taking place on-line, the recovery technique involves a Roll File dump of the task in question for off-line debugging. All other systems require a Log Transaction tape for recovery of data records and adequate restart. The Transaction Log tape produced by ENVIRON/1 is a complete log of all transactions entered into the system, and provides an automatic and complete audit trail for error recovery purposes. It contains the master record before the transaction, the detailed transaction, and the updated master record. The level of detailed information stored on the Transaction Log is determined at run time by the user via a control card. The context area of the Roll File is also used in restart procedures since it provides the last checkpoint of context for each active terminal. Checkpointing of selected blocks is provided thru use of a special command.

In the event of an application program check, the task and its related blocks are dumped to the Roll File for later printing; a message is sent to the affected terminal alerting the operator to the fact that his previous input was not satisfactorily processed. None of the other terminals are aware of any failure. The number of operator retries for a message is specified on a control card and failing that, the operator must sign-on to the application again. This type of terminal restart will probably be the most common form of error recovery within the system since application programs are the most volatile item in the software.

If, for any reason, a partition restart is necessary, any updated records are placed in their original state with the before copy on the Log tape so that data integrity is maintained. The entire process is automatic with the possible exception of re-entering the last terminal message.

A system failure is handled in much the same fashion. If the Transaction Log tape unloads as in a power failure, the operator may be required to reposition it (an I/O error on the last incomplete record should indicate the end of processing).

By logging the after copies of each record, ENVIRON/1 has made the recovery of a damaged disk pack much easier. Rather than falling back to a previous backup and rerunning all transactions against the file again, which may introduce timing differences and hence, inconsistent results with the real-time system; the file may be recovered by reconstruction with the after copies of these records ensuring that customer A does not receive customer B's order and vice versa, a faster and a safer process.

These two levels of restart and data recovery reflect the NEVIRON/1 philosophy that system availability is paramount to most online users. ENVIRON/1 attempts to provide debugging aids to prevent future problems once an error condition has occurred, i.e., Roll File dumps, etc. It is also realized that an online system will never be entirely error-free so that a means must be provided to recover from these errors with a minimal impact on users. By providing both a spatial redundancy (writing records to files and the Transaction Log) and a time redundancy (audit trail), ENVIRON/1 attempts to recover from most, if not all, user-encountered problems.

Testing – Terminal Simulation Facility

Since in many cases, a user does not have ready access to terminals for testing purposes, especially when an existing real-time system is being converted; ENVIRON/1 provides a facility for testing the on-line system through terminal simulation in the background partition. This facility requires only two tape drives and a computer configuration as close as possible to the desired ENVIRON/1 configuration. One input tape contains actual or planned terminal messages in their projected format while the other tape is the ENVIRON/1 Transaction Log output tape. The capability of logging all input messages, all data reads, and all program reads makes this output very helpful in debugging. Since each output block is time-stamped, a complete audit trail of each message through the system is given.

These tapes are run against an ENVIRON/1 system hopefully containing an exact replica of the real-time system with the omission of the actual device dependent modules. The resulting system is driven by the input tape as fast as the time interval allows it to be handled; which may be harder than the actual terminals would drive the system. The simulated system will execute until a program check occurs, at which time a dump of all related blocks will be made to the Roll File.

Since other terminals may be simulated at the same time, the run may continue to end of tape or be stopped at the first failure. Following termination of the test run, the Roll File may be printed as well as the log tape for aids in debugging. The Roll File provides a snapshot of the data and context blocks at the time of failure. The Log tape may be selectively printed by type of data or time interval to give a history of a portion of the test run. Statistics are also provided for the user on the activity of the entire system including number of accesses to each program page, number of times data was found inbuffer store and the number of times the system dropped into the wait state. The average response time and message activity by terminal and access activity by data set is given at the end of each log tape so that the user has an indication of the overall system performance during testing.

Measurement

A very important and highly sophisticated feature of ENVIRON/1 is the System Measurement and application testing facility. ENVIRON/1 compiles and maintains statistics on various aspects of hardware/software system performance. This measurement facility provides the means for the user to evaluate system performance and growth potential. To make system evaluations possible, the ENVIRON/1 Control Program maintains a statistics on I/O channel utilization, message traffic per terminal, buffer storage usage, program page statistics, and the number of concurrent tasks in execution. These statistics are compiled by terminal and by data set as well as for the system as a whole. Measurement provides the means for fine-tuning a system. It also allows future growth to be predicted realistically and bottlenecks to be seen before they occur, thus giving management visibility of their system's current and future performance.

The systems measurement and testing facility of ENVIRON/1 allows the user to actually determine his projected system performance in terms of his current system input. This facility permits an actual Transaction Log tape to be input to the current or a projected system configuration. The log tape is used rather than an artificial test input tape to more closely match the actual system's peaks and valleys on a minute-by-minute basis. Since the tape is time-stamped, the ability to play it against the system at varying speeds is obtained. Statistics generated on device, channel, cpu, and program performance will indicate possible bottlenecks to the user. The tuning capability of ENVIRON/1 may also be used to vary the mix of program versus data pages, number of concurrent tasks, number of multiple I/O requests, polling interval

and partition or region size by changing a control card.

Since response time is usually a good measure of system performance, the average response time may be plotted as in Figure 12. The response time may be expected to remain nearly constant until a bottleneck (resource overload) begins to occur. Then the response time will rise rapidly for very small increases in system volume.

The determination of the point at which an overload occurs is important not only to obtain current system capacity but to identify the cause of the overload and alert planners to future system needs. This cause may be deduced through an analysis of the statistics given with each run and possible solutions tested by varying the tuning parameters. For instance, if an indexed data set appears particularly active it may be desireable to provide more data buffer store to eliminate some I/O activity or move the file to another pack, or even move a portion of the file such as the indices. If a particular set of program pages is highly used, it may be possible to combine some of the pages or to provide more program buffer store to reduce I/O activity on the Program Data Set.

The system measurement and testing facility of ENVIRON/1 is provided for better performance today as well as future growth. It permits the user to determine what his system is doing today through daily Transaction Log statistics and allows him to prevent system bottlenecks before they occur by driving the system at tomorrow's message rates.

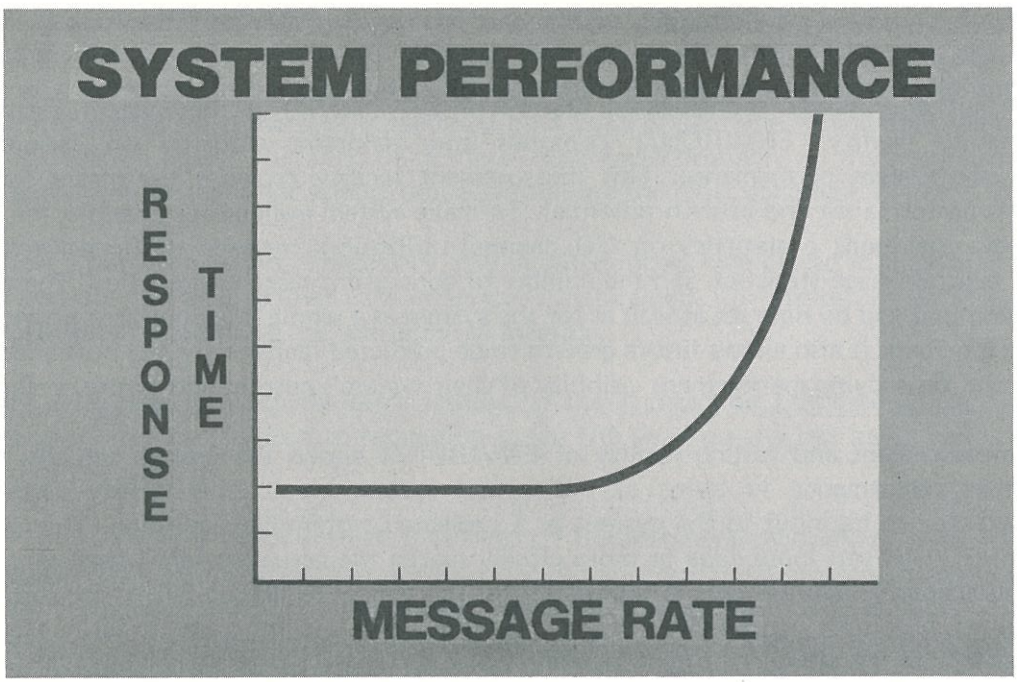

FIGURE 12

occurs within a task, the system will dump the task in error, but will continue with the other tasks in the system. If there is damage to the ENVIRON/1 control program, the system will effect restoration of the control program without stopping system operation. If the operating system (OS or DOS) fails, a quick restart can be obtained with little or no operator involvement.

The adaptability of ENVIRON/1 is provided through device independence, operating system independence, terminal and configuration independence and in general, the complete independence of the software from external considerations. The ability of the system to adapt to changing circumstances with minimal impact and reprogramming provides the user with easy maintenance. Since any device dependence within the system is modular; it is easily changed, removed or added to.

ENVIRON/1's adaptability limits the impact on application programs from data format changes. With ENVIRON/1's ability to adapt to any hardware/software environment, the impact of environmental changes on programs and data is significantly reduced, if not totally eliminated.

ENVIRON/1 also provides adaptability from an operational viewpoint. The system is insensitive to operational changes; i.e., if devices on the system become inoperative or are detached from the system, ENVIRON/1 will facilitate adaptation to a new system configuration. The additional storage device independence provided by ENVIRON/1 allows data sets to be spread across different storage device types and further permits redistribution of data sets as the need arises.

ENVIRON/1 permits the user to change and expand equipment configurations and data structures with no impact on existing programs. ENVIRON/1 delivers high performance, without reprogramming, for the entire model range of IBM System 360-370 CPU's from a Model 25 with 32K partition supporting a few terminals up through a Model 195 with thousands of terminals. The ENVIRON/1 user can also convert his on-line system from DOS to OS/360, with no reprogramming required, not even recompiling, if object decks are available. This is a very important feature, which can save many man-hours when system expansion becomes necessary.

The resultant system affects savings in development and maintenance time as well as reductions in hardware expenditures through CPU and core and I/O optimization techniques employed in ENVIRON/1. By providing a system easy to implement, ENVIRON/1 enables the user to effectively reduce the cost of system installation and operation. The compiler and DOS/OS compatibility allow future conversions to be easily accomplished with a minimum of effort.

In summary, ENVIRON/1 provides a real-time control package which efficiently manages resources and the environment for the user; allowing the user to focus his attention on his application rather than the complexities of on-line data processing. The software system is designed for ease of use and future growth as well as high performance. Significant features of ENVIRON/1 include:

- Buffer Store Concept
- Paging
- Dynamic Core Allocation
- Virtual Addressability
- Multi Requests of I/O
- Rollin/Rollout
- Single Station Concept
- System Restart & Recovery
- Complete Transaction Logging
- Multi-tasking
- Multi-threading
- Storage Device Independence
- Improved Access Methods
- System Measurement & Tuning
- System Simulation & Testing
- System Integrity & Availability
- High-Level Compiler Language
- Application-Mainline Loop Concept