M A N A G I N G   D A T A   E N T R Y
W I T H
N O N - S T A N D A R D   T E R M I N A L S

INTRODUCTION

Solutions to general problems are often derived from specialized solutions of unique problems. In this case, our search for a generalized solution to the problems of handling non-standard (non-HP) terminals in an HP environment began with a need to interface Lear Siegler ADM-1 terminals to an HP 3000. Our client's motive was purely financial: he had already purchased the terminals for use with another computer system, and did not want to spend money twice for the same equipment.

Because of the commercial nature of the application, COBOL was chosen for the implementation language. We found that the language facilities in COBOL were at least adequate for all required operations except terminal communications.

PROBLEM STATEMENT

The problems involved with using standard COBOL input/output statements to perform terminal I/O are well known. However, a brief review of these problems may prove instructive.

FIXED LENGTH I/O

Terminal I/O can be approached within standard COBOL language constructs in two ways: the field-oriented ACCEPT and DISPLAY verbs and the file-oriented READ and WRITE verbs. In either case, the length of the character string to be input or output is defined by the length of the buffer or field and is fixed at compile time. On output, while it is theoretically possible to put out exactly the number of characters desired at any time, the mechanism required to allow the output length to be computed at execution time involves defining a field or record containing the desired number of characters and then writing the code to select the proper field or record. This is potentially cumbersome to write and inefficient to execute.

Using the READ verb for input allows termination of input by a carriage return and recognition of an end-of-file condition. However, if it is desired to have the computer "take the input away" after a certain number of characters have been input, it is necessary to define not just records but entire files with varying record lengths.

Use of the READ and WRITE verbs also allows more flexibility in interfacing with the operating system, because they are connected with SELECT statements which can be linked to MPE FILE commands to override system default parameters.

The recent enhancement of the ACCEPT verb to allow variable length input is a vast improvement over the old method, which required one to always input the maximum number of characters allowed. However, there are still many drawbacks to the use of ACCEPT. Besides the loss of flexibility in terms of MPE, there are significant editing deficiencies, which are indicated below.

EDITING

COBOL string manipulation leaves much to be desired. Such operations as point alignment, numeric editing, date checking, and the like, are handled very inefficiently.

CONTROL CHARACTERS

All ASCII terminals recognize a number of "non-printing" control characters. With a CRT terminal, the number of functions controlled by these characters can be very large. Most such terminals have a number of functions in common: clearing the display, cursor positioning, definition of unprotected fields, and the like. While these functions are ubiquitous and essential to the "human engineering" of applications, they can be extremely difficult to encode in a COBOL data division. COBOL does not have the ability to define one-byte octal constants. Not all terminals have a "display functions" capability. Some terminals like the ADM-1 will not even transmit all control characters from the keyboard to the computer: they treat all escape functions as local, even if the terminal is in remote mode.

In order to persuade the COBOL system to transmit control characters, it was necessary to translate the octal values of those characters into decimal, define computational fields with the decimal value, and redefine the computational fields as character fields. At that point they would be output without further translation by the system. Needless to say, the results were clumsy and unreliable. Programs were difficult to maintain, and the prospect of supporting another type of terminal--with different escape sequences to perform the same functions--was unsettling if not downright frightening.

SOLUTION APPROACH

It was against this background that our requirements for a terminal interface evolved.

We chose SPL as the implementation language because:

* Variable length terminal I/O was easily and efficiently implemented;

* Translation tables could be placed in the code segment, making it efficient to place the interface routine in the SL;

* A simple interface to COBOL could be designed;

* Within the context of the defined task, the code could be made self-documenting and easily maintainable;

* The gain in execution speed and compactness of object code was too great to ignore.

In order to address the problems stated above in a comprehensive way, the following design objectives were established:

* Control functions should be able to be specified without using the actual control characters;

* Application program logic should not have to change to support different kinds of terminals;

* The communication buffer should be defined as a character string so that COBOL could conveniently access various portions of the buffer;

* The system should operate with reasonable efficiency.

After the interface was completed, and we implemented the application system, we were able to support fourteen terminals in 256KB of memory with no more than 1.5 second transaction response time. This was about what we had predicted, but we have since learned that other users with similar loads have experienced much worse performance.

ANALYSIS

While we have not analyzed fully the reasons for this comparatively good performance, an understanding of HP 3000 architecture would indicate that the following factors are important.

CHARACTER MODE

By necessity, our system had to operate exclusively in character mode, because an ADM-1 cannot communicate with an HP 3000 in block mode. This turns out to be a good choice for any terminal, because the architecture of the HP 3000 is such that character mode I/O is handled more efficiently than block mode. There is greater randomness in the distribution of the CPU work load and in channel operations, allowing the MPE dispatcher to schedule better.

Character mode allows editing to be done on a field-by-field basis, with edit errors displayed immediately upon the operator's hitting carriage return. Since control can be returned to the application program after every field is input, extended editing such as file verification can also be performed immediately, further distributing the CPU work load. Furthermore, if a portion of the input screen is not required for certain options, it does not have to be read at all, further saving CPU and channel time.

With character mode, information can be displayed upon (or erased from) the screen at any time, thus allowing such "human engineering" conveniences as displaying a name associated with a file code, interrupting normal dialogue to ask a special question, redisplaying a numeric field after point alignment and zero fill, and the like.

In this sort of environment, the computer is perceived by the operator as being very responsive. As is well known, this perception has a positive influence on operator acceptance of new systems.

MEMORY MANAGEMENT

Among the many factors that influence response time, stack size and code segment size are two of the most important. If either one grows too large, thrashing becomes a problem.

Thrashing is more heavily influenced by stack size than by code segment size for two reasons:

First, code segments are shared among all processes using them, thus allowing some overlapping memory use from one process to another. If two processes are running the same program, the overlap is complete: all code segments are shared. However, there is at least one data stack per user per process. Stacks are not shared, and the larger the stack required by a program, the more rapidly the main memory approaches saturation as the number of users increases.

Second, data segments must be saved when they are swapped. While a code segment may be simply overlaid, a data segment must be copied to disc before it is overwritten by the incoming segment. Thus, it takes at least twice as much time to swap a data segment as it does to swap a code segment.

One aspect of block mode communication disciplines not discussed until now is the memory requirement. It is always the case that buffer space must be provided to handle the maximum number of characters that could be transmitted at once. In the case of block mode transfers, this could be 2000 characters or more. With HP VIEW/3000, the memory requirements are even greater. It is not unusual to have this system add 6000 bytes to each user stack, not to mention the extra data segments created by its use of KSAM. Needless to say, an environment with thirty users active with VIEW could be using 180,000 bytes of stack space just for the VIEW buffers.

In contrast, a system employing character mode can be designed to operate on a field-by-field basis. We determined that the maximum field size that we would handle should be 80 bytes, the length of one line on many CRT's. This allowed us to define the complete interface buffer, including control fields and the 80-byte data area, in an area only 140 bytes long. The result is a significant reduction of stack space. Those same thirty users could be using only 4,200 bytes of stack space for the terminal buffers.

As with VIEW and other system packages, our interface routine resides in the system SL so that there is only one code segment allocated, no matter how many programs are concurrently using the interface.

CONCLUSIONS

Using character mode, it is possible to realize significant improvements in response time through reduction of memory requirements and improvement in distribution of CPU and channel activity.

Using an SPL interface allows the introduction of standard edit capabilities and variable length I/O to a COBOL program with a minimum of difficulty. This SPL interface also reduces the need for the application programmer to be concerned about the specific terminal type that the program may be communicating with.

Because all asynchronous terminals capable of ASCII communication with an HP 3000 must at least be capable of character mode line discipline, the use of such an SPL interface allows the HP 3000 user to conveniently interface terminals that are not manufactured by HP, possibly resulting in significant cost savings.