HANDLING A LARGE (200 PLUS) NUMBER TERMINALS ON THE HP-3000

by N.M. Demos

SYNOPSIS

I Introduction

It is possible within certain constraints to handle a large number of terminals on the HP-3000. The terminals must be dedicated. The application must not consume any undue amount of computer resource per terminal. An efficient monitor program must be employed to manage the system resources required. Application code must be written to interface effectively to the monitor and to process efficiently. In addition, as the number of terminals increases, a way must be found to reduce the overhead implicit in Hewlett Packard's present terminal handling hardware and software (the asynchronous terminal controller and multi-point terminal software with the SSLC). While these constraints may be violated in minor ways, it must be remembered that the HP-3000 has a finite computing and file access capablility. Applications requiring a high degree of computing and a large number of file accesses for each entry from a terminal are not good candidates for the approach presented here.

In given applications, specialized hardware may have to be interfaced to the HP-3000. It is not the purpose of this presentation to emphasize or advocate a particular hardware approach, although one hardware solution will be presented in the context of the software approach described.

It is the objective of this presentation to explain the programming techniques employed to support a large number of terminals.

A case study of an existing implementation will be used to present some of the considerations and requirements in handling a large number of terminals on a HP-3000.

II History

In mid 1976, the company, with which this author is associated, started discussions on the possibility of implementing an on-line Point of Sale system. As the company involved was an O.E.M. for Hewlett Packard the problem was considered in the context of using a Hewlett Packard Computer. While the company had substantial expertise on the HP-1000 series of equipment it was decided to propose to the customer that an HP-3000 be used as the main frame, because the customer wished to do other applications on the computer system. As a result of these discussions, the Hewlett Packard 3000, with communications equipment and a Point of Sale terminal designed by the company and their associate engineers, was proposed. The Point of Sale

terminal consisted of a logically integrated RS-232 display terminal and a 40 character wide printer of our design, with a connection for a cash drawer which was opened by a special character.

The Point of Sale terminal was connected to an intelligent, remote controller which acted as a control and buffering device, so that data could be transmitted synchronously over leased telephone lines. An SDLC-like protocol was implemented for the communications network so that maximum efficiency could be obtained. Multi-drop capability was included so that a number of remote controllers could share one line. At the host sight, the front end was designed to support the remote controllers. In order to interface to the HP-3000, with minimum disruption to the 3000 hardware and operating system, the front end was made to look almost like the HP programmable controller. This was the product offered at the time by Hewlett Packard for communication between a 3000 and the HP-1000 line of computers (using the 2100 series as the CPU component). We were therefore able to use the universal interface driver supplied as part of the MPE operating system with only slight modification. The universal interface (the same as the line printer controller) on the HP-3000 is a parallel interface. It supports programmed versus direct I/O so that CPU interference is much less than with the asynchronous terminal controller, which employs direct I/O.

We now had all the hardware and system software components required to make the system operate. Next we had to design and implement a software monitor to acheive our goals.

III Implementation of the Software Monitor

At the time of the design of the software monitor to achieve our objective of supporting a large number of terminals on the 3000, we had to keep in mind several subsidiary objectives. These were:

A. We wished to maintain the integrity of MPE.

B. Time to implement and economy of coding was desirable.

C. Response time at the terminal should be in the one to two second range; it should not exceed six seconds.

D. Memory utilization should be minimized.

E. As other terminals were to be attached to the system through the asynchronous terminal controller, the monitor must relinquish control when it is not performing useful activity.

The HP-3000 environment with its MPE operating system, imposed an environment that included the following characteristics:

A. Although I/O no-wait was implemented, there was no way to determine if an I/O operation was complete and do any processing if it was not.

B. There was no way for the monitor (which was the father process) to suspend and be awakened by EITHER I/O completion or an activation by a son process (meaning that its task was complete and it had data for the monitor).

In this environment, several decisions were made, as follows:

A. Because of the required efficiency, the monitor and the application code would be programmed in SPL.

B. The monitor would handle all input and output to the front end, but handle no other I/O directly (except communication with the monitor terminal, that would be used to control the communication subsystem).

C. Each terminal would be assigned its own process; this process would do no input-output of its own.

D. The Image Data Base Management System would be used for the files where random access was required.

E. There would be separate, asynchronously running processes for Image access and other disc access (the latter required

mostly for a Transaction Logging file).

F. All interprocess communication would be through extra Data Segments.

G. All transfers of control would have to be explicitly handled through the father process.

H. Queues would be set up and maintained for the Image and Disc Access processes so that the monitor could perform other activities while an individual terminal might be waiting for Image or Disc Access.

With these decisions in place a main process loop was designed, that started with a read to the front end and ended with an IOWAIT if no activity was present in the subsystem. The monitor runs in the BS queue so that, at this point (IOWAIT) other sessions could perform activity. Also, whenever there was activity and a process was started, the monitor would suspend when no data was available from the front end.

Approximately four months after this system was designed the hardware and software were brought up for final test purposes. With some minor adjustments, (mostly involved in debugging firmware in the front end and controllers) the system worked satisfactorily. About six weeks later, it was put into production.

The orginal design goal was to support eighty terminals. The next required step was to expand to support 100-120 terminals. This was accomplished through implementation of an additional front end to lessen the load on the first front end, and some changes in the monitor program. The major change was the implementation of read and write queues (buffers) for the front end I/O. This was necessary, not only to smooth momentary peak loads, but also because the front ends had limited buffer space. For technical reasons, it was necessary to read all the data possible from the front end ("drain" them) before attempting to write to them. The implementation of write buffers allowed processing to continue, even though there was a momentary peak load on the out-put side of the system. These changes, an increase in memory size, and other minor fine tuning adjustments, allowed us to increase capacity as required at that time.

The next required step was to expand to support 100-120 terminals. This was accomplished through implementation of an additional front end to lessen the load on the first front end, and some changes in the monitor program. The major change was the implementation of read and write queues (buffers) for the front end I/O. This was necessary, not only to smooth momentary peak loads, but also because the front ends had limited buffer space. For technical reasons, it was necessary to read all the data possible from the front end ("drain" them) before attempting to write to them. The implementation of write buffers allowed processing to continue, even though there was a momentary peak load on the out-put side of the system. These changes, an increase in memory size, and other minor fine tuning adjustments, allowed us to increase capacity as required at that time.

The next requirement was to support up to 200 terminals doing the same application. At this point it was realized that the process for each terminal approach would not suffice. First, each process requires a MINIMUM overhead of 2048 bytes, for its stack, not considering working storage for data movement and calculations. Secondly, research indicated that even without paging to and from disk, it required an absolute minimum of eight milliseconds for a monitor to activate a process, suspend and then to have the son process reactivate the father. This figure holds with no processing being doing in the son. For these two reasons, it was decided to discontinue the process per terminal approach and use a procedure of the monitor for application code. A place to store the data required for each terminal and a method of ascertaining where in the code, processing for any given terminal was to continue, were required. In other words, some of the housekeeping that MPE performed automatically through process handling had to be simulated by other means.

Because Q-relative storage could be used for data required between input-output requests, the only requirement was for the holding of data required between I/O requests. This was accomplished by assigning 70-word blocks in -DB. These were managed by the monitor. Because the application required that additional, considerably more dynamic, memory be allocated to store print lines for a given transaction, a method was devised to manipulate the original Q location upward and use the available space as temporary storage. This was also handled by the monitor, although individual procedures could request data and release date in this area directly.

The other requirement was handled satisfactorily, if somewhat inelegantly, by calling a special exit sub-routine before leaving the procedure. This special sub-routine stored its own exit address in the special data area mentioned above. In complementary fashion, a start sub-routine always called at the

beginning of the procedure, retrieved this exit address and used it at its own exit address. Some additional housekeeping was required, but this is essentially how the procedure operated. This required a coding discipline where no input/output could be done in a procedure or sub-routine called by this application procedure. For obvious reasons, it was also desirable to keep the data required between input/output operations to an absolute minimum. In general these were the only programming constraints necessary to implement this approach.

Because it would have taken considerable recoding to make the disc handling process a procedure and because access to it was not required often enough to have a significant impact on performance, it was left as a separate process. Because there was no way of doing no-wait I/O with the Image files, it was necessary to leave the Image code as a separate, independently running process. This change was implemented and results were as expected. Thrashing was eliminated and response time was improved with 160 terminals active. In this application environment and the way this system operates, it is felt that 200 terminals operating at once would be feasable.

The next design goal would be to support in the neighborhood of 300 terminals. We will soon be doing simulations to see where the present system has bottle necks and where we might make improvements. Depending upon the results of our analysis, we will probably make one or more of the following changes to improve performance:

A. Restructure the Image Data Base.

B. Take the random processing out of Image and use a custom random access subsystem. Use extra Data Segments for more queue capacity for the Image and disc access queues.

C. Review and restructure the main process loop in the monitor to make sure that the most likely event is tested for first and that tests are made by groups. At the same time we would review and if possible refine our methods for terminal address calculations, etc.

D. Based on simulation tests, make sure that the most critical resource has the highest priority when it comes to a question of what task the monitor should undertake next.

At this point we realize we are pushing the limits of the HP-3000 and we will be looking for ways to offload some of the processing. We are looking currently at two ways of doing this. The first one involves linking one or more HP-3000s together via a DS-link and splitting the loads between them. The second approach involves putting a mini-computer at high volume locations with capability to handle individual transactions without interrupting the host 3000. Currently the latter approach is favored. Our design specifies that the mini at the location will have its own data base that will be updated during off hours. Also all programs, as well as data, will be down loaded from the host system. This latter approach is favored, because it will result in satisfactory backup in case either the line or the host 3000 fails.

Although the system presented here may be more exotic then some users can foresee implementing, it has fulfilled its purpose and has performed with a larger throughput than anyone had originally envisioned. System uptime has been more than satisfactory, and processing has been very inexpensive in its environment. We feel confident in recommending that any situation requiring a large number of dedicated terminals where the application processing is not too extensive, be programmed using the above approach. The key criteria are the requirement for a large number of dedicated terminals where each input from a CRT terminal does not require a significant amount of computational or disc processing. Under these conditions an excellent response time will be achieved, with economic hardware and software costs.

N.M. Demos