Programming For Multiterminal Application
A User Experience Managing Processes and Data Segments

Michael A. Casteel
Computing Capabilities
Mountain View, CA.

1. Introduction

This paper describes the approach used and experience with an HP3000 data entry/inquiry application which supports an unusually large number of terminals. Developed in the summer of 1976, this application was to be based on COBOL, provide short response times and support 36 terminals initially, with reasonable capacity for growth. That these requirements have been met is now apparent: the system now hosts about 75 block mode terminals on 5 ATCS, providing typical response times of two to three seconds. Main memory is still the original 384K bytes, while a second 47M byte disc has been added to accomodate growth of the master file to over 100,000 records.

This gratifying result is the product of an excellent match between the application and the facilities of the HP3000 system. It is certain that not every application could support 75 terminals and still provide acceptable response time, especially in 384K bytes. Still, the application system presented here, which takes full advantage of the Process and Data Segment management and terminal I/O facilities of the HP3000, has not yet reached the limit of its capacity. The system has operated about 20 hours per day, six days per week for three years with very little down time due to software or HP hardware problems.

2. The Application

The application is online data entry and inquiry for work-in-process (WIP) inventory tracking in semiconductor component manufacturing. The inventory master file resides on the HP3000, where it is queried and updated from CRT terminals using any of 10 interactive COBOL programs and 4 common subprograms. Almost all file maintenance activity takes place on-line on the HP3000. The entire master file is copied to tape every night, and processed on a large mainframe for accounting and reporting.

WIP inventory is tracked in Lots, each Lot representing a group of components to be processed through a sequence of manufacturing operations. The master file holds one record for each Lot containing space to record the result of up to 30 operations. The 10 transaction types available permit Lot records to be added, deleted, or updated, as well as providing a number of ancillary functions such as interdepartmental transfers.

The most common transaction (80% of transaction volume) is inventory movement update. Due to the structure of the Lot records, this transaction need only update the selected record with the result of the latest operation. Access to edit files is rarely required, and there are no logical relationships or chains to be maintained in the master file. The master file is standard MPE file accessed by custom File Manager routines using calculated direct addressing by Lot number (see Section 3).

Clearly, user disc file I/O is light in this application. Terminal I/O is another matter, as the typical transaction involves displaying the contents of an entire Lot record,

2. The Application (continued)

then reading the data back in a block mode transmission after the terminal user has keyed in updates.

3. Disc File Access

As described in Section 2, user disc file activity is close to an absolute minimum for the typical transaction in this system. This is indeed a key factor in the performance and terminal capacity of the system. While this was recognized early in the design, two factors were given special attention: a highly efficient storage/access technique and protection from simultaneous update requests.

On-line access to the master file is random using a key containing the Lot number to be accessed. Logically, the master file corresponds to a stand-alone IMAGE Master Data Set. However, the System designers chose not to use IMAGE for this application due to the relatively high overhead (including separate Data Base Control Blocks) associated with IMAGE and the simple file structure needed for this application. A simple File Manager was designed instead, consisting of a number of SPL subroutines to provide read, write and update access by key to records stored in a standard MPE file. Disc storage requirements were minimized by including specialized record compression and segmentation logic, so that most records could be stored in a single 256-byte disc record. Main memory space and file system overhead were both minimized using NOBUF access to eliminate MPE buffers.

The question of simultaneous update was of great concern to the system designers. While there is little chance that two users would try to update the same Lot record at the same

3. Disc File Access (continued)

time, such a possibility had to be accounted for in the design of the system. The desired approach was to use a "record locking" facility: When a user calls a record to the screen for updating, the record should be "locked" to prevent another user from calling up the same record. After the record has been updated, the lock should be released to permit other users to access the record. Since at the time HP offered only file (or Data Base) locking, the File Manager included a record locking mechanism using a shared Extra Data Segment for a Record Lock Table.

When an application program requests a record for possible update, the File Manager checks the Record Lock Table. If the record key is found in the table, the request is rejected and the user informed that the record is currently in use. If the record key is not in the table, it is inserted and the record itself returned to the application program. The key is removed from the table automatically when the record is rewritten, or by a special File Manager function if no update is required.

The File Manager routines access the Record Lock Table using the DMOVIN and DMOVOUT intrinsics. The table itself is protected from simultaneous update using a RIN: The File Manager is careful to Lock the RIN before and Unlock it after accessing the Lock Table. Thus, while a record may be locked for many minutes, no user need ever wait long to access a record or to discover that the desired record is being updated by another user.

3. Disc File Access (continued)

The use of the Extra Data Segment facility for record locking placed a significant restraint on the processing structure of this application: since Extra Data Segments cannot be shared between sessions, the terminal users could not log on as MPE sessions in the usual manner. All users must operate within a single job/session. As shown in Section 5, use of a single job/session was desirable for this application in any event, but the restriction still prevents separate update jobs from running while online processing takes place. The new IMAGE locking facility removes this constraint for IMAGE users, but the restriction on sharing Extra Data Segments remains one of the more obvious deficiencies in MPE.

The custom File Manager code for this application occupies about 1K words, some 16 pages of SPL source. No Privileged Mode operations were used, only the Data Segment Handling (DS) special capability.

The nightly task of copying the master file to tape presented one more challenge: rapid sequential processing of the 120000-sector master file. The small, one-sector blocks on the master file are optimum for random on-line processing using NOBUF access. Normal sequential reads, however, transfer at most one block per revolution of the disc, a considerable overhead for 120,000 blocks. For this reason, the copy program uses Multi-Record (MR) access to read many blocks in each transfer. MR access transfers as many full blocks as possible in a single disc read or write, up to one full track. Adjacent blocks are transferred as one, provided that each block completely fills one or more disc sectors, i.e. the block size must be a multiple of 128 words. If the block size is not a multiple of 128 words, MR access transfers

3. Disc File Access (continued)

only one block at a time to or from the disc. This application uses an SPL subroutine to read up to one full track of 128-word records into the DL-to-DB area and return one record at a time to the COBOL copy program.

4. Terminal I/O Handling

This application uses block mode transmission with formatted CRT screens. A major design concern was the possibility of overloading the HP3000 with 2400 baud block mode terminal I/O, with resulting loss of data or poor performance. When this application was designed, the HP3000 Series II was expected to require 300 micro-seconds of CPU time to process each character interrupt on the Asynchronous Terminal Controller. At 2400 baud, the terminal sends one character every 4 milliseconds. A simple calculation shows that no more than 12 terminals can transmit at one time without overloading the computer, a situation resulting in a "data overrun" error (loss of a data character) on one or more terminals.

MPE uses the HP264X DC2/DC1 protocol in an attempt to prevent data overruns. When the user presses the ENTER key to initiate a block transmission, the terminal sends only a single character: DC2. The ATC hardware can retain this one character until the interrupt is serviced, without incurring a data loss for this terminal. When MPE receives this character, it recognizes it as a request for block transmission (if Terminal Type is 10 for this ATC port). The transmission will not take place until MPE sends a DC1 character to the terminal. MPE takes advantage of this protocol to hold off the transmission until the number of other terminals actively transmitting or receiving drops below the level of potential overload.

4. Terminal I/O Handling (continued)

WhenHP264X terminals are used, it should be noted that the user's program must process the DC2 character, because the cursor must be positioned to the first data field to be transmitted before the DC1 trigger is sent. This is the reason for FCONTROL #29, to enable user control of block mode transfers. The DC1 trigger is sent by MPE every time a read is issued to a terminal, so the next FREAD will trigger the transmission.

The system described here does not use HP terminals, but uses Lear Siegler terminals which were already owned by the organization. These terminals offered two features which were used to advantage: The terminal automatically positions the cursor for a block mode transmission, and it allows for data compression when transmitting to the computer. The first feature relieves the HP3000 from the task of positioning the cursor whenever a DC2 is received, thus saving the additional FWRITE and FREAD required for an HP264X. The data compression feature effectively suppresses transmission of trailing spaces in any unprotected field. On the average, this significantly reduces the number of characters transmitted and thereby the load on the computer due to terminal input. In return, the program must restructure the input using field separator characters and knowledge of the screen format.

Screen formatting, terminal I/O and data compression/expansion are performed in this system by a set of Terminal Handler routines. The Terminal Handler is coded in SPL, and provides the COBOL application programs with a high-level interface akin to HP's DEL. Only numeric edits are provided, protected fields are supported (display only), and only one program

4. Terminal I/O Handling (continued)

call need be coded in a typical program to write data (using a named screen format) and wait for input. Forms are catalogued in an external library using a form maintenance utility.

In order to take advantage of the DC2/DC1 handshake supported by MPE, the terminals used in this system were modified to observe the HP264X protocol. Unfortunately, data overruns (FCHECK error 28) are still observed in about 10% of all transmissions. These errors are recovered automatically in the Terminal Handler software by forcing retransmission with an escape code sequence.

The custom Terminal Handler software occupies about 1.5K words, 18 pages of SPL source. No special capabilities were needed for terminal handling.

5. Application System Structure and Process Management

The user terminals in this system are not used to run sessions, but are instead opened as files by the application system. This means the users never log on to MPE; they cannot, since their terminals are not configured to accept jobs or sessions. This approach not only saves the effort of training users to type HELLO, but affords an extra degree of system security by preventing access to MPE commands and utilities. System resources are conserved as well: every session initiated from a terminal requires the allocation of a process for the MPE Command Interpreter and Virtual Memory for the Command Interpreter's stack. This process and stack is in addition to that of the application program.

5. Application System Structure and Process Management (cont.)

This system instead uses a single master program initiated at the beginning of the day, which creates a process for every user terminal in the system using the CREATE and ACTIVATE intrinsics. This is effectively the same as issuing a "RUN" command for each terminal, but does not involve the Command Interpreter. Figure 1 illustrates the resulting process structure. Remember, because there is only one job in the system, the application programs cannot use DISPLAY and ACCEPT to communicate with the terminal user; DISPLAY and ACCEPT refer only to the job's $STDIN and $STDLIST for all processes in the job. The Terminal Handler instead provides access to the user terminal by opening a file using the Logical Device Number of the terminal.

The master process also allocates and initializes the Extra Data Segment for the Record Lock Table and allocates the local RIN used to control access to that table.

Each created process is assigned to a particular terminal by Logical Device Number, and each runs the same program containing all the application code. The application main program, written in SPL, starts by opening all disc files used in the application and opening a file to access the specified terminal. Once opened, the terminal file must be conditioned for use by a sequence of file control intrinsic calls. This system uses:

```
FOPEN(,%604,  =CCtl, Undefined ASCII records, NEW
      %404,  =NOBUF, Read/Write
      -2400, =Maximum Record Length
       LDEV) =Logical Device Number
```

5. Application System Structure and Process Management (cont.)

```
FCONTROL #37   =Allocate a terminal
FCONTROL #25   =Set RS as input terminator
FSETMODE #6    =Suppress CR/LF on input
FCONTROL #13   =Echo off
```

Finally, each process gains access to the record lock table by a GETDSEG intrinsic call using the same ID the master process used in allocating the Extra Data Segment. The RIN is automatically usable without issuing another GETLOCRIN.

Each process stores global control information such as MPE file numbers returned from FOPEN and the Data Segment index returned from GETDSEG in the DL-to-DB area of its stack so it is accessible to the File Manager and Terminal Handler procedures at any time.

Once this initialization is complete, the SPL main program begins application processing by presenting a menu screen on the user's terminal. When the terminal user selects function from the menu and enters a valid password, the COBOL program implementing the selected function is called as a subprogram. The COBOL program interacts with the user by calling Terminal Handler subroutines, and accesses disc files by calling the File Manager subroutines. Processing overhead is minimized since all needed files were opened when the system was started.

Stack space for each terminal is minimized by using the $CONTROL DYNAMIC compiler option on the COBOL subprograms. This option defers allocation of Data Division storage until the subprogram is called, and releases the space when the subprogram ends with GOBACK, to return to the main program.

5. Application System Structure and Process Management (cont.)

The $CONTROL SUBPROGRAM option on the other hand permanently allocates stack space for every such subprogram, which is clearly undersirable for this application. In a further effort to minimize memory requirements, the main program shrinks the stack on return from an application subprogram, using the ZSIZE intrinsic. When the next subprogram is called, MPE automatically expands the stack as needed.

6. Performance Measurements

The performance of this application system was observed in actual operation by monitoring terminal I/O. Code was added to the Terminal Handler to collect CPU time and wall-clock time elapsed between the completion of a read (input) and the initiation of the next write (response). The accumulated times and the number of input/response pairs was recorded separately for each user by application function. Figure 2 is a graph of the combined statistics for all users and functions as observed during a peak day of operation with 55 terminals.

A total of 39, 144 responses were recorded in the 14 hours observed , for an overall average of 2747 responses per hour. A peak observed rate of over 3800 responses per hour was observed over two hours (14:30 - 16:30). After 17:30 the rate remained around 2200 responses per hour.

The overall computer response time (wall-clock) averaged 430 ms/response, while the longest observed response took 11.9 seconds. Average CPU time was about half the average response time.

6. Performance Measurements (continued)

Due to the method of observation, this time does not include data transmission to or from the terminal. However, since the average number of characters received on input was observed to be under 200, data transmission time can be estimated at around 2 seconds at 2400 baud. Thus, the average total response time is estimated at under 2.5 seconds.

This analysis does not account for two variables which could effect (lengthen) the actual response time seen by a terminal user: First, there could be a delay after the user presses the ENTER key before the terminal begins data transmission to the computer. This can result if MPE holds off the transmission because of simultaneous block reads and writes already in process. Second, there could be a delay in dispatching the application process once the read from the terminal is complete, for example, if the process has to be swapped in before it can continue to execute. First-hand observation of user terminals indicates that neither of these factors contributes significantly to typical response time.

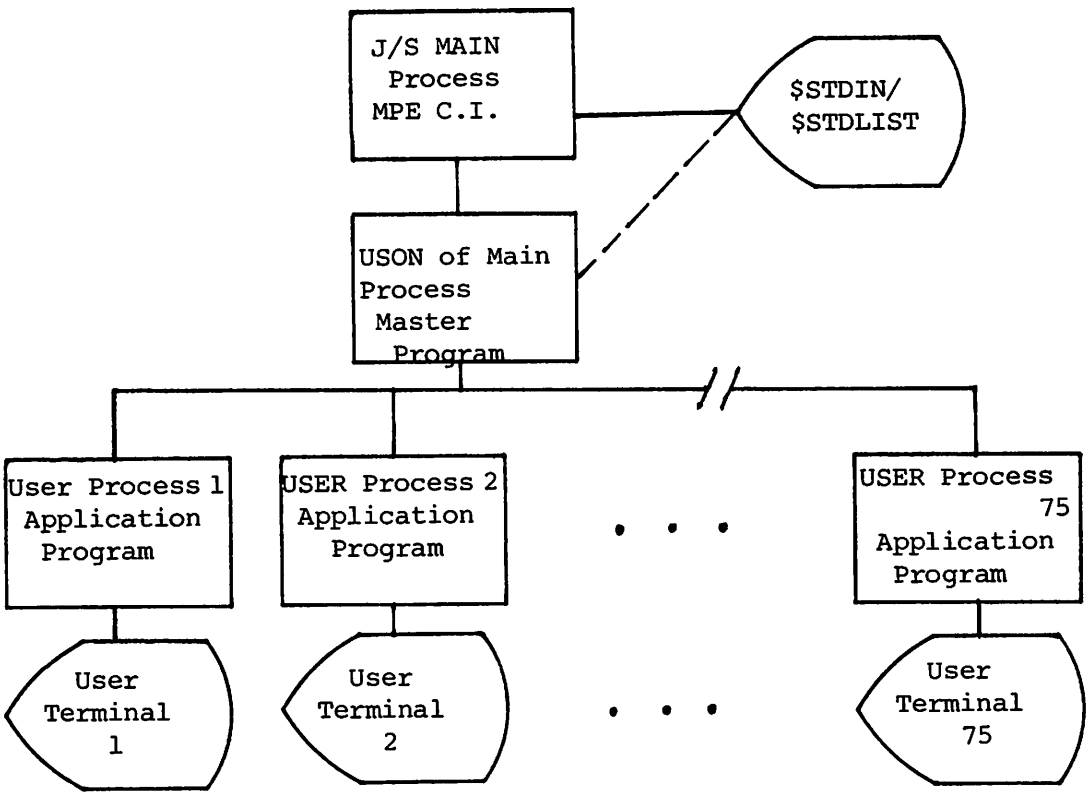

Figure 1 - Process Structure

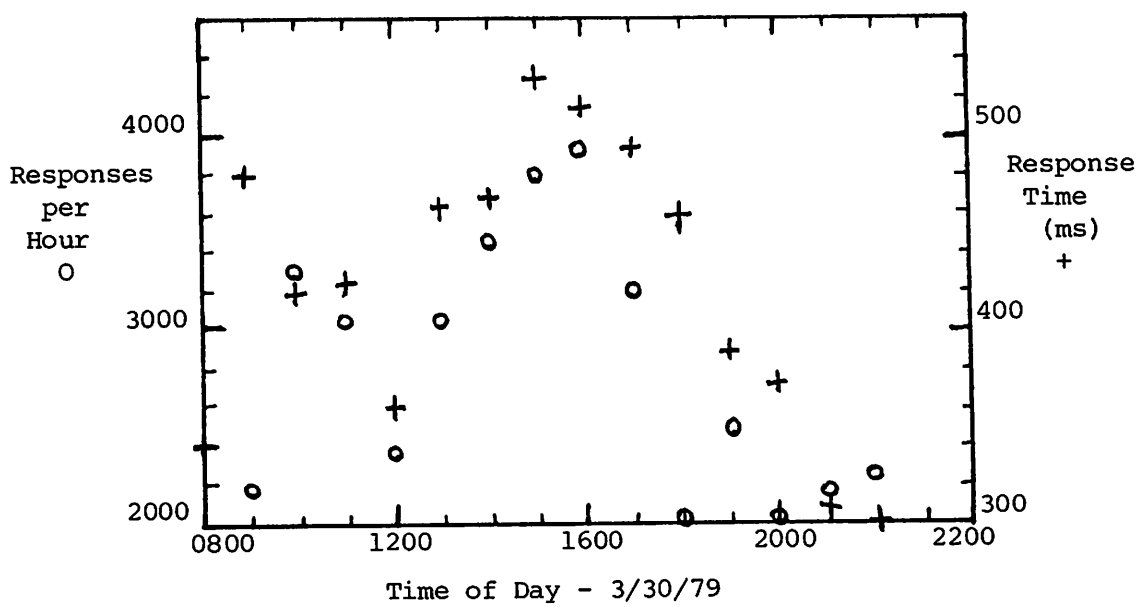

Figure 2 - Response Times and Rates