First Quarter 1981
Vol. IV, No. 1

Journal

HP General Systems Users Group

CONTENTS

SPOTLIGHT

Image Locking and Application Design 3
Gerald W. Davidson

FEATURE ARTICLES

System Performance and Optimization Techniques for the HP/3000 6
John Hulme

News About Pascal on the HP/3000 11
David J. Greer

Indexing Bryn Mawr's First Dean: An Essay In Data Entry and Text Formatting 15
Jay Martin Anderson

In the Beginning 17
Marc Covitt

CONTRIBUTED LIBRARY CORNER

By Mark Wysoski 18

TIPS AND TECHNIQUES

Set Up Ideas for Local Users Groups 19
Marc Covitt

RETURN CARD

Bug/Enhancement Poll 21

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

The 1981 **JOURNAL** Publication Schedule, as given below, includes the deadlines for receipt in the Executive Office of articles submitted for publication. Also given are the months of publication.

HPGSUG JOURNAL PUBLICATION SCHEDULE

Article Submission Date	Quarterly Issue	Month of Publication
October 15	First	January
January 15	Second	April
April 15	Third	July
July 15	Fourth	October

Contributions: HPGSUG Executive Offices

IMAGE LOCKING AND APPLICATION DESIGN

Gerald W. Davison
9700 South Cass Avenue
Argonne National Laboratory
Argonne, Illinois 60439

When designing IMAGE-based applications it is quite natural to design the database as the first step. However, when the application is complex and the number of on-line users will be large, the designers should go immediately into a second iteration of database design before designing the application programs. This second iteration should involve defining how locking will be enforced in the application.

Once large-scale program design, and worse yet, programming, is underway, the process of refining the locking discipline will become an unexpected and expensive exercise. A well-though-out locking discipline will return benefits in reduced programming and testing efforts. It will also simplify your explanation to the users who may have a hard time understanding what locking is or what it has to do with the application in the first place. And once in production, it is never pleasant to explain to the users that the strange occurrences of degraded responses and aborts are really locking difficulties.

IMAGE LOCKING The concept of locking as implemented in IMAGE is fairly simple, and Hewlett-Packard does a good job of explaining the concept in the IMAGE manual. A brief summary is provided here for later reference.

IMAGE can lock at three levels:

- The whole database.
- One dataset.
- A specific data entry within a dataset.

The calling program can request one of two lock types:

- Unconditional (program waits until lock completes).
- Conditional (IMAGE returns immediately and tells the session if the lock succeeded).

The calling session can have one of two capabilities when locking:

- One lock before an unlock is required.
- Multiple locks before an unlock is required (requires MR capability for the session).

If the database is opened in access mode 1, which is highly desirable when forming a locking discipline,

IMAGE enforces the following locking rules:

- No locks are required for reading even if another session has the data locked (locking may be desirable to ensure consistent data).
- To modify a data entry using DBPUT, DBDELETE, or DBUPDATE requires a look at the database, dataset or data entry level.
- To add or delete a data entry in a manual master dataset using DBPUT or DBDELETE requires a lock of the entire dataset.
- IMAGE does not require the program to lock associated automatic masters when a detail dataset entry is added or deleted.

Several important things to remember about

IMAGE locking are:

- Successful locks on the same locking set queue behind the current active lock causing all requesting sessions to suspend.
- One DBUNLOCK removes ALL locks for the requesting session on the specified database.
- All programs must lock on the same data item if data entry level locking is used.
- A program can control locks and unlocks on multiple databases independently.
- And, of course, when using multiple locks per session all deadlocks require a halting of the system in order to clear the deadlock for the sessions involved.

LOCKING DISCIPLINE CONSIDERATIONS

Assuming that the application definition has enough detail to design an initial database, how does one design the locking discipline to go along with it? Since the application is assumed to be complex with many on-line users, it is very likely that the database design and transactions will also be complex. The following initial goals will help to ensure a locking discipline that is flexible and one that minimizes situations leading to poor user response due to locking:

- All sessions will open the database(s) in access mode 1 to ensure enforcement of the locking discipline.

— All sessions will be given multiple lock capability via the MR capability; multiple locks are required for complex transactions on complex databases.

— If the application will allow, the usage of the database locking level should be forbidden when on-line sessions are executing (some batch processes may use database locking to keep all other conflicting users out).

— Since dataset locking prevents all other sessions from updating the locked dataset, the usage of the dataset locking level should be kept to an absolute minimum when on-line sessions are executing.

— To avoid lengthy session waits on data locked for extended periods by other sessions, use conditional locking whenever it is possible and unconditional locking when required.

At this point the definition of the locking discipline becomes application dependent. Several concepts are very helpful when defining the locking discipline for an application with dozens or scores of datasets and scores, if not hundreds, of programs:

— Key lockable entities.

— Bidding for access to key lockable entities.

— Bidding priorities.

— Terminal user think time.

In the discussion of key lockable entities, consider an application that integrates the purchasing, receiving, and accounting functions of a large organization. Instead of thinking of scores of lockable entities, one per dataset, it is beneficial to think of key entities such as requisitions, purchase orders, receipts, and invoices. These broader concepts can cover a large portion of the datasets in the application.

In the preceding example a particular purchase order is a key lockable entity which a unique terminal session can bid for via locking at a data entry level. These entities can be requested or bid for via conditional locking as soon as the entry is identified to ensure that no other session has the desired data associated with the purchase order. If the particular entity is busy, no great harm is done if the terminal user is told to retry later. If the bid succeeds, then the session can proceed with the transaction.

The idea of priority of bidding on key lockable entities is also very important to the locking discipline. Using an example of a terminal session bidding on a receipt associated with a purchase order, the session bids conditionally on the higher priority purchase order number before locking on the lower priority receipt associated with that purchase order.

The concept of terminal user think time (when the user goes to lunch in the middle of the transaction) is essential to the definition of the locking discipline. Disregard for think time can cause severe user response degradation if key entities are held by one session for extended periods of time. Typically, these key entities are of great interest to many sessions working on totally different functions or the same function but with a different key value. Examples could be one session undating vendor information at the same time a session creating a purchase order is trying to update vendor totals. Or two sessions trying to create a purchase order and both trying to get at the next purchase order number to be assigned. This situation is troublesome due to the absense in IMAGE of the ability to selectively unlock specific locked entities. It is an all or none situation when using DBUNLOCK. Even though the situation is difficult, the locking discipline requirement is very clear. **(DO NOT LOCK THESE ENTITIES OVER TERMINAL USER THINK TIME!)**

DATABASE AND APPLICATION DESIGN CONSIDERATIONS Once the locking discipline has been thought out initially, the designer should consider the two locking constraints imposed by IMAGE that affect database and application program design. Briefly stated, these considerations are:

— The need to lock manual master datasets in their entirety for any add or delete.

— The automatic removal of ALL locks on a given database when DBUNLOCK is issued.

The first consideration involves the basic design of hashed keys and synonyms in IMAGE. It appears unlikely that it will ever change. The second consideration involves the option to selectively unlock entities, and this should be on every IMAGE user's list of highest priority items to be changed by Hewlett-Packard.

The need to lock entire manual master datasets for an add or delete suggests changes to the database design. Since it is highly probable that key locking entities will be held for extended periods of time, including across terminal user think time, it

is desirable that these entities reside in detail datasets and not in manual masters to avoid conflicts with sessions that must lock entire manual masters.

A general solution to the problem of locking an entire manual master dataset is to keep the usage of manual master datasets to an absolute minimum by using only detail datasets with automatic masters. This requires one extra disk access to retrieve the data initially or to add or delete it. However, the usage of detail datasets will simplify the design and reduce the probability of locking difficulties.

The need to unlock all locked entities before a complex transaction is complete usually is the result of a desire to free an entity which is of general use to many sessions as quickly as possible. One solution is to require the application program to set an in-process flag at the highest priority key locking entity associated with the transaction. This will allow the session to retain control of the transaction data without another session claiming it during the unlocked period. This solution is dangerous since session aborts will leave the in-process flag set. This will cause the key locking entity, such as a purchase order and all of its associated data, to be unavailable to any session until the in-process flag is cleared.

A secondary solution would be to move these high-usage entities to another database which would allow the session to unlock only those entities that other sessions may need. Although secondary databases have associated overhead, the simplified application program design and reduced code to relock may be substantial considerations.

CONCLUSION Although the design of a locking discipline may seem to be a secondary concern when designing your application database and programs, several IMAGE considerations and the need to provide reliable programs with good user response should place this task very early in the design. In order to be effective, the design of the application locking discipline must take place in concert with the database design.

FEATURE ARTICLES

SYSTEM PERFORMANCE AND OPTIMIZATION TECHNIQUES FOR THE HP/3000

John E. Hulme
Applied Cybernetics, Inc.

INTRODUCTION The purpose of this paper is to introduce the reader to certain techniques which can improve system performance, throughput, and run-time efficiency on HP/3000 computers. These improvements will typically reduce response time and increase data processing productivity.

This paper will not simply tell you what to do and what not to do. In many cases there are trade-offs involved and it is more important to understand the principles behind the techniques than the techniques themselves. And because analogies often help us to learn by giving us a new perspective, we will make use of a non-data-processing illustration.

SOME BASIC PRINCIPLES The first thing to understand is that any given computer can execute a finite number of instructions in a fixed amount of time. When that theoretical limit is reached, no amount of tuning can "squeeze" extra instructions into the computer. For the most part, however, computers do not bog down because we ask them to do too much, but rather because we cause them to trip over themselves in the process of doing it.

This leads to the second important principle: At any moment the computer is either (1) doing productive work, (2) getting ready to do productive work, or (3) waiting on some external action before it can proceed with productive work. As a program is initiated, thereby causing a certain sequence of instructions to be executed, we will call the execution of those instructions "productive work". Whether the "productive work" is really necessary or not, and whether it is efficiently or inefficiently organized, are issues to be addressed later. But a more significant fact of computer life is that usually only a small percentage of the computer's time is spent executing application program instructions.

A CRUDE MODEL To illustrate these principles, imagine a "library for the blind". The librarian sits behind the desk waiting for a blind person to walk into the library. This is the "waiting period". When the blind person arrives, the "getting ready" period begins. The blind person tells the librarian which book to retrieve and by one method or another the book is retrieved. The librarian now begins the "productive work" phase, reading to the blind person from the selected book. When the reading is completed, the librarian may return the book to the shelf or leave it on the desk. Then a new waiting period begins.

If the library is a busy one, we can imagine that one or more assistants might be hired to transport the books between the librarian's desk and the book shelves. Let's imagine that there is one assistant for each wing of the library. The librarian can do more PRODUCTIVE WORK (reading to the patrons), spending less time GETTING READY (still looking things up in the card catalog, but now dealing with the assistants instead of transporting books). A new type of waiting is introduced, however: waiting for assistants to bring books back.

In this analogy, the librarian represents the computer's central processing unit (CPU), by which all the productive work is accomplished. Like our imaginary library, the HP/3000 has only one CPU. To improve throughput we must maximize the CPU's productive time.

Each patron represents a log-on session or job. The librarian's desk represents the computer's main memory. It is of a limited size, merely a workspace, in comparison to the stacks of book shelves which correspond to the mass storage devices. Finally, each assistant represents an i/o channel transferring data to and from disc, for example.

While illustrating some important concepts, this analogy does not accurately model the run-time environment of the HP/3000, or any other computer. How could we refine the model to make it more realistic?

THE MODEL REFINED At the risk of distorting the human situation, let me suggest four refinements which make our model more nearly resemble the actual computer processes:

1. The "library" should be regarded as a collection of
   (a) read only instruction manuals and reference tables (programs and constants) and
   (b) numerous loose leaf volumes (files) containing sheets of current figures and data (records) which may be periodically replaced, revised, removed, or added to.
2. The "librarian's" job should be generalized to include any type of service that can be per-

formed on the basis of pre-printed instructions and supplied data.

3. The computer always deals with a COPY of whatever is stored on the disc, and usually just a few records at a time. So let's imagine that instead of asking a library assistant to fetch a particular book, the librarian will specify a limited number of paragraphs or data sheets and will ask the assistant to bring a photocopy of the desired paragraphs (colored paper for instructions; white paper for data).
4. Because the processing speeds of a computer are so great, our model operates in slow-motion by comparison. Allowing that the librarian can do in one hour what an HP/3000 can do in one second (i.e., using the scale of one hour for each second), the assistant could handle 20 to 60 requests per hour, and the equivalent of a 60-word-per-minute typist could enter one character every 12 minutes. A 2400-baud rate would be equivalent to a maximum of 5 characters transmitted per minute, and a 600-line-per-minute printer would correspond to one line every 6 minutes.

SLOW-MOTION PERFORMANCE SIMULATION

Visualize this scenario from the patron's point-of view (refer to figure 1): You walk into the library, find an empty cubicle (terminal), and make yourself comfortable. You begin to formulate and transmit your library card number and password (log-on) at the rate of no more than 5 characters per hour. (If it will relieve the agony, you may imagine that you spend the time drawing very large, very elaborate block letters). Depending on the facilities available in the cubicle, you will either transmit each letter as it is formulated or accumulate several characters (maybe even hundreds) and transmit them in a burst. In either case, you transmit each letter separately by ringing a bell, and, when you have the librarian's attention, holding up the card with the letter on it. The librarian records each character of your message on a notepad corresponding to your cubicle, then continues with his other business. Finally you send a character which means "that's the end of what I'm sending you".

The librarian eventually verifies that you are a qualified user of the library and sends you back a standard message which allows you to proceed. This process may require the librarian to send his assistant to the book shelves several times, e.g., to get a procedures manual, index of users, table of passwords, welcome message, etc.

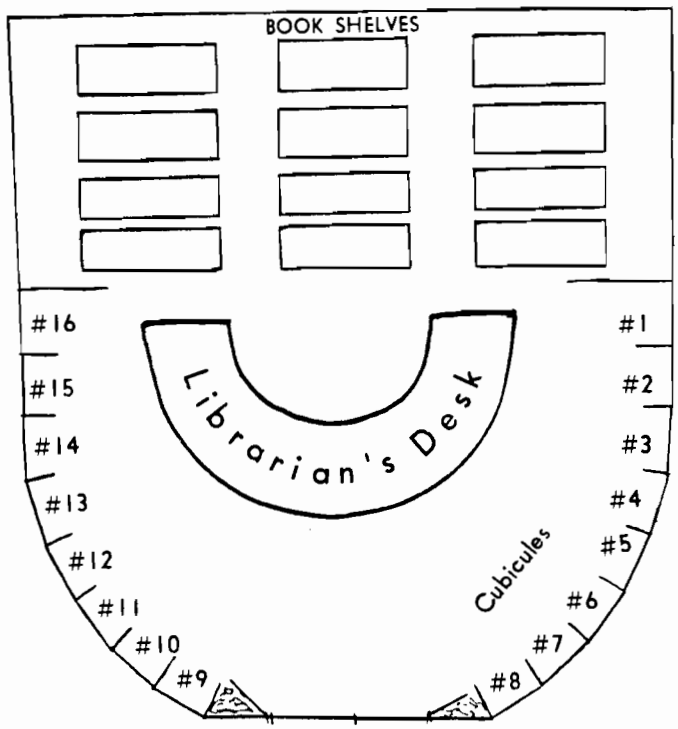


Figure 1. The Library

Next, you painstakingly tell the librarian the name of an instruction manual (program) you want him to follow in performing some service for you. He has the assistant get him a copy of the first paragraph (segment) of the instruction manual (unless a copy happens to be sitting somewhere on the desk already). He also gets a copy, your own personal copy, of a worksheet (your date stack) associated with the specific instruction manual you have specified.

In case there is not enough empty space on the desk for these papers, the librarian first clears some space by either (a) throwing away one of the instruction sheets, (b) having his assistant put the worksheet for some other patron in a special holding file (virtual memory), or (c) having his assistant take one of the data sheets back to the loose-leaf it was copied from and replace the original with the new version.

The librarian now goes to work following the instructions you have requested. This will continue until (a) he comes to a point in the instructions which specifies he is to send certain information to you and/or ask you for additional input; (b) he comes to the end of the page or is otherwise instructed to refer to another page, one which is not currently on the desk; (c) the instructions require that information be fetched from the book shelves, taken there to be filed, or sent to some output device; (d) a predefined length of time elapses (a 500 microsecond quantum corresponds to one-half hour in our

model); or (e) the librarian completes his assignment and disposes of your worksheet.

In any of these cases, the librarian will go back to work for one of the other patrons, provided he has all the resources necessary to do so. If not, he will wait (until the necessary information is fetched by the assistant or transmitted by one of the patrons). Depending on what you've asked the librarian to do, and how busy he is doing things for the other patrons, it may take hours or even days before he gets back to you. But then again, it may take days for you to formulate the equivalent of one screen of input, too (at the rate of 5 characters per hour).

THROUGH THE EYES OF THE CPU Now let's reverse roles and look at the situation from the librarian's perspective. Try to imagine yourself as a calm, unemotional, purely methodical being who is never responsible for mistakes because he does precisely as he is told. You couldn't care less if someone gets poor response time; you aren't to blame, because you only rest when there's nothing for you to do. In fact, you purposely set things aside during peak demand periods to do in your spare time. But you can't take credit for that either — you're only following directions from the MPE handbook.

2:08:17 Ring! There's the bell in cubicle five. He's holding up the letter "R". Write it down on memo pad #5 (line buffer).

2:08:20 Here's the library assistant with the record session #12 requested. Oops! The worksheet for session #12 has been set aside (swapped out to the system disc). Send the assistant for it and wait a minute.

2:08:24 A ring from cubicle #8. That's a carriage return. Time to reinitiate session #8. Make a note to send the assistant for the worksheet when he gets back.

2:08:29 Wait some.

2:09:00 Wait some more.

2:09:16 Oh good, something to do (the observer's feelings, not yours). A ring from cubicle #3. A "7". Write it down.

2:09:20 Here's the assistant. Put worksheet #12 on the desk. Send him back for worksheet #8 — no, there's not room for it. Give him the worksheet for session #5 and send him to file it (we're waiting for input from cubicle #5). We'll send him for worksheet #8 next time.

2:09:24 Okay, now to get to work on task #12. First set the timer for 30 minutes. Now add I to J and put the result in K.

2:09:28 Move W6 to W2. Move . . . hold it, there's another ring from #3. Say, that's only a few seconds . . . must be a block-mode terminal. Write down the "9" and go back to work. Move X to Y. Call the procedure "XFORM". Oh, it's on the desk already — it hardly ever gets thrown out, that's because nearly every program uses it.

2:09:40 Another ring from cubicle #. This time it's a minus sign. Continue with "XFORM". Convert the first letter of Y to upper-case. Now the second letter. Now the third. Now the fourth. That's all. Return to the main program. It's still in memory. Move the new Y to F3.

2:09:52 Another ring from cubicle #3. A field separator. Resume task #12. Perform FLAG-SET subroutine. It's in another segment, one that's not in memory. Make a note to send for it. Suspend task #12 for a minute.

2:10:04 Cubicle #3 again. Just a blank, but write it down anyway. That's "7-9-minus-field separator-space", so far.

2:10:14 The assistant has finished filing worksheet #5. Send him now for worksheet #8.

2:10:16 Cubicle #3. Another space.

2:10:19 Interrupt from the printer saying the last line has printed successfully. Now reactivate the spooler job — it's instructions are still on the desk and so is the buffer containing the print-line. Initiate i/o transfer.

2:10:26 2-second wait.

2:10:28 Cubicle #3. A third space.
12-second wait.

2:10:40 Cubicle #3. A fourth space.
12-second wait.

2:10:52 Cubicle 3. A fifth space.
12-second wait.

2:11:04 Cubicle #3. A field-separator.
5-second wait.

2:11:09 Worksheet #8 is here. Send assistant to get a copy of FLAG-SET routine.

Now to process this input from cubicle #8.
Edit first field. OK. Edit second field. OK. Move first field to R1.

2:11:16 Cubicle #3. The letter "H".
Move second field to K2. Edit third field. Isn't numeric but should be. Transfer to error handler in same segment.

2:11:29 Cubicle #3. The letter "O".
Prepare output to tell cubicle #8 about error. Comment: It's a shame, but since he's in block-mode, he'll have to retransmit the whole screen again, after correcting the error in field 3. And who is to say other errors might not be detected after that? And you, the librarian, can receive those 873 characters, one every 12 seconds for nearly three hours. But you don't mind. It's only a job.

2:11:40 Cubicle #3. The letter "V".
Finish putting error message in the output buffer. Initiate transfer to cubicle #8. Mark task #8 eligible to be swapped out.

2:11:47 Cubicle #11. The letter "P".

2:11:52 Cubicle #3. The letter "E".
FLAG-SET routine is here. Continue with task #12. Move 1 to FLAG. Add 1 to COUNT. Exit back to mainline. What! The assistant had to fetch a separate segment just so we could do that?

2:11:59 Cubicle #11. Oh, oh. Two block-mode devices transmitting at once! Record the letter "I".

2:12:04 Cubicle #3. The letter "R".

Comment: Stop, I've had enough of dinging bells! This place sounds like a hotel lobby, not a library!

OBSERVATIONS As this analogy indicates, there are three factors which reduce overall system performance:

a. unnecessary disc i/o (most serious),
b. unnecessary terminal i/o (too common), and
c. unnecessary CPU usage (rarely the problem in an on-line environment).

EXCESSIVE DISC I/O The primary cause of excessive disc i/o is INADEQUATE MAIN MEMORY to hold the required work space (stack and data segments) for each concurrent process, plus all frequently referenced program segments, plus a reasonable mix of infrequently referenced program segments.

The HP/3000 is very good at handling multiple concurrent users, even when they won't all fit in memory together. In fact, the use of virtual memory, combined with a well-designed algorithm for selecting which segment to overlay, allows the system to operate efficiently even in cases where a single program exceeds the limits of main memory.

The thing to remember, however, is that code segments put a relatively small load on the system while data segments put a potentially disastrous load on the system. In the first place, code segments can be split up and made as small as the programmer wants them to be. Secondly, they do not have to be rewritten to virtual memory when the main memory space is to be re-used: they are simply overlaid. Data segments, on the other hand, tend to expand, and can be split only with difficulty. Since their contents may change, they must be rewritten each time the process is swapped out, and reread each time it is swapped back in. Finally, whatever data space is required must be repeated for each process that is active. Therefore, if you are supporting 20 terminals, any reduction in data requirements would produce 40 times the benefit that an equivalent reduction in code requirements would produce.

Aside from upgrading to a large machine, a shortage of main memory can be averted by:

a. reducing the number of concurrent processes (not an attractive option),
b. reducing the average stack or data segment size,
c. reducing the size of the average program segment,
d. organizing program segments better so that out-of-segment transfers occur less often to non-resident segments and so that often-used code is collected in compact segments that are likely to stay in memory, or
e. some combination of the above.

When adequate main memory is available, swapping is unnecessary, and disc accesses (which are very expensive in terms of time) will be expended strictly for data retrieval and storage. Once swapping begins, the computer's "productive" activities are at the mercy of "waiting". In the worst case, "threshing" occurs, which means that every time a session gets a turn at execution, either the program segment has been overlaid or the session's work space has been swapped out.

It is worth noting that the use of IMAGE (or of KSAM) causes the allocation of extra data segments. Specifically, each IMAGE data base that is open requires a data segment large enough to hold one copy of the root file plus four complete data base buffers. If a program accesses multiple data bases, or if the root file or buffers are large, the memory requirements will be substantial, and with many terminals running data base applications, the memory requirements can add up very quickly. Granted, the advantages of using a powerful access method may outweigh the costs of additional memory demands, but such tools should be used carefully and not indiscriminantly.

It should also be noted that the use of block-mode requires extensive buffers in the stack (at least as large as the largest screen to be transmitted). The use of VIEW/3000 may add another 6000 bytes of buffer in each user's stack, not to mention the extra data segments created by its use of KSAM. If you have 20 users, this amounts to 120K extra bytes of memory or more.

EXCESSIVE TERMINAL I/O Major causes of excessive terminal i/o include the following:

a. Transmitting unnecessary characters (trailing spaces, leading zeroes, insignificant digits, etc.) to the computer, a necessary consequence of fixed-format or block-mode input.

b. Transmitting the same data to the computer more than once, as occurs in block-mode when a whole screen is retransmitted to correct an error in a single field.

c. Retransmitting to the computer data which has not been changed since it was received from the computer. This too is the result of block-mode transmission.

d. Repeatedly displaying prompts at the terminal instead of using protected background forms.

Since each character of input consumes critical resources, every effort should be made to ensure that only significant data is transmitted (no extraneous zeroes or spaces and only those fields that are new or have been modified).

It is not only wasteful of computer power, but also destructive of operator morale, to wait until a whole screen of data has been entered and transmitted to the computer before discovering that the screen is invalid due to a duplicate key or an unrecognized search-item value, etc.

It is equally inefficient (for the computer, that is) to display a screen of data, have the operator update a single value and transmit the whole screen back to the computer. In an extreme case, this could amount to over a thousand characters transmitted just to change one or two characters.

EXCESSIVE CPU USAGE Besides the costly i/o overhead, it is altogether possible that a retransmitted screen will be completely re-edited, values packed and unpacked, and fields reformatted even though only a single field was updated, and maybe even if NOTHING was updated. This is one cause of unnecessary CPU usage.

Most editing and reformatting done in COBOL subroutines requires excess usage to begin with, and it is far better to allow such work to be done in SPL subroutines, where it can be done efficiently. Including such subroutines in the COBOL programs also causes bulkier segments, which is likely to increase the need for swapping. The best solution is to incorporate all editing within the terminal-handling module itself, since it is already being shared by all on-line programs and is therefore likely to remain constantly in main memory. There are a multitude of factors which can unnecessarily increase the so-called "productive work" which the CPU has to do. Because computers are seldom CPU-bound in an on-line environment, few people exert the effort to truly optimize CPU performance anymore. Whenever it is a problem, more careful analysis of the program(s) in question will usually yield a more efficient method of solving the application problem.

Often, more careful analysis will also yield a better solution from the point of view of disc i/o as well, both in terms of swapping, code-segment switching, and data retrieval and storage. One word of warning, however: more efficient solutions (CPU-wise) are very often more complex, and to the extent that they increase stack space, or code-segment size, or they require more transfers from one code-segment to another, they may prove counter-productive.

One situation in which heavy CPU usage can be very detrimental is when on-line processes are competing with batch applications for CPU resources. This can be vividly illustrated by running a COBOL compile, an Editor GATHER ALL, a sort, or the BASIC interpreter at the same time on-line programs are running. Block-mode applications exhibit many of these same tendencies and can severely impede response-time for character-mode applications when both types are running concurrently.

SPECIFIC OPTIMIZATION TECHNIQUES

1. Re-segment programs so that no segment exceeds %5000 words.
2. Set the blockmax parameter on IMAGE schemas as low as possible.

3. Use extra data segments where possible and free them up when finished, rather than increasing stack space for large temporary buffers.
4. Don't keep files open unnecessarily.
5. Don't abuse IMAGE:
   a. eliminate sorted chains where possible.
   b. carefully evaluate tradeoffs of increasing or eliminating secondary paths in detail data sets.
   c. use "@;" or at least "*;" for item lists wherever possible.
   d. only use binary keys (in master file) when overlapping keys can be avoided.
   e. don't let synonym chains get very long.
   f. when loading master data sets, store only primaries on the first pass, making a second pass for secondaries.
   g. keep master data sets less than 85% filled.
   h. periodically reorganize detail data sets that have long chains associated with a frequently-accessed path (puts consecutive records in the same physical block).
   i. keep the number of data sets in a data base as small as practical without requiring many programs to open multiple data bases.
   j. keep IMAGE record lengths to a minimum.
6. Have operators exit programs when not in use.
7. Use a field-oriented terminal handler which performs standard edits for you.
8. Use formatted screens with protected background whenever the application is appropriate to such use.
9. Keep terminal i/o buffers small; if possible, eliminate block-mode i/o altogether. (Don't use block-mode and character-mode i/o at the same time.)
10. Don't use VIEW without a lot of memory.
11. Don't use DEL at all.
12. Run CPU-intensive jobs (including compiles, preps, and Editor GATHER ALL) when on-line applications are not running, or at least run them in a lower-priority sub-queue.
13. Set the system quantum for a shorter period than recommended in the MPE manual (but don't overdo it — some experimentation may be necessary).

NEWS ABOUT PASCAL ON THE HP/3000

David J. Greer
Robelle Consulting Ltd.
#130-5421 10th Ave.
Delta, B.C.
V4M 3T9

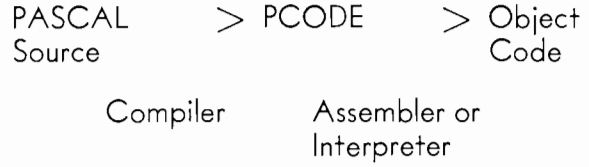
Recently, there has been much talk about the programming language PASCAL. This article discusses which PASCAL compilers are available on the HP/3000, what kind of state these compilers are in, and where PASCAL is going on the HP/3000.

HISTORY PASCAL was developed by Niklaus Wirth in Zurich. The principal aims were to provide: (1) a language in which structured concepts could be taught easily; and, (2) a language that would be relatively easy to implement on many machines.

Both these original aims have direct application to business and scientific programming. The language supports structures which emphasize structured coding concepts. Programs written in PASCAL tend to be structured, which in turn makes them easier to maintain and understand. Because STANDARD PASCAL has been implemented on many different machines, including the HP/3000, it is a good vehicle for writing portable software.

PORTABLE PASCAL P4 In order to make PASCAL available on many machines, Urs Ammann, K. Nori, Ch. Jacobi, K. Jensen and H. Nageli wrote the portable PASCAL-P4 compiler, which is written in PASCAL itself. Approximately 80% of the PASCAL compilers in the world are based on this compiler.

The P4 complier takes PASCAL source code and compiles it into symbolic code for a hypothetical stack machine called a "P-machine". The individual implementer of PASCAL writes an assembler or interpreter for the "pcode". Later, the original compiler source is changed to generate the host machine's object code directly. The following is a schematic description of how pcode works.



HP/3000 PASCAL-P4 In the HP/3000 contributed library, there is a version of PASCAL developed by Grant Munsey, Jeff Eastman and Bob Fraley. This

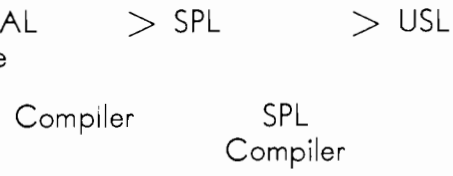
(continued on page 19)

PASCAL 2.7-V

HISTORY PASCAL 2.7-V is another version of PASCAL for the HP/3000, developed in Vancouver as a project in compiler design at the University of British Columbia. This compiler is a modified version of the one that is available in the contributed library.

MAJOR PROBLEMS Two of the problems in using the Contributed library PASCAL are that it is somewhat difficult to specify all of the command sequences for invoking the compiler (although this has been fixed with UDCs), and the compiler is very slow. Both of these problems stem from the long process involved for getting from PASCAL source code to USL files.

PASCAL 2.7-V does away with the assembly stage of the compilation process. Instead PASCAL source code is translated directly into SPL code. The PASCAL compiler then invokes the SPL compiler to produce the USL file.



By eliminating the pcode stage of the process, 20-40% savings in elapsed compilation time were realized. Also, by having all of the compilation stage in the compiler, it was easier to use the compiler. The SPL stage of the compilation process was not eliminated because it was too difficult to work with USL files directly.

MINOR PROBLEMS The original contributed compiler did not print error messages. The compiler now prints a summary of all of the error numbers which occurred during compilation, along with their associated error messages. Also, a compiler option has been provided which causes all PASCAL reserved words to be underlined. This greatly enhances the readability of PASCAL programs.

ATHENA COMPATIBILITY With the Athena MIT release of MPE (2011), all Pascal programs on the HP/3000 ceased to work, including all of the contributed versions of Pascal. The cause of this was that, as of the Athena release of MPE, Qinitial was two words higher in the stack. Since Pascal made certain assumptions about where Qinitial would be in the stack, Pascal programs stopped working. Pascal 2.7-V fixes this bug by making no assumption about where Qinitial should be.

STANDARD PASCAL There is a large effort world wide to provide a comprehensive standard for PASCAL. Any compiler which claims to compile STANDARD PASCAL must compile all parts of the defined standard correctly. In order to help implementers and users of PASCAL find out whether their particular compiler meets the standard, A. Sale and R. Freak have written a suite of PASCAL programs to test PASCAL compilers.

The suite consists of approximately 300 PASCAL programs. Each program is compiled and executed by the PASCAL compiler. The results of each program are analyzed for errors. If a compiler meets STANDARD PASCAL completely, then there will be no errors from any program in the suite.

PASCAL 2.4-V was tested using the suite. It had approximately as many errors as other compilers based on the original P4 compiler. This is to say that the compiler did about average. Several errors were fixed with PASCAL 2.5-V and later versions.

USAGE My version of PASCAL is currently is use at about a dozen installations around the world. Many installations are using Pascal-S (see Appendix I) for teaching PASCAL. Also there are two installations using PASCAL to convert software from other machines to the HP/3000.

AVAILABILITY Because we need a PASCAL compiler to develop programs now at Robelle Consulting, I have been and will be maintaining this PASCAL Compiler. This is not a supported Robelle product; but, for $100 U.S. (to defray costs), we will send a 1600 BPI copy of the latest version to interested users ($200 if we have to send a bill or convert the tape to 800 BPI). As of November 1980, the latest release was VERSION 2.7-V (bugs fixed regarding large VALUE parameters, programs with only one simple global variable and production of the SPL listing; ARRAY OF FILE is now supported). The previous version 2.6-V provided support for the ATHENA release of MPE. Inquiries can be directed to me at Robelle Consulting Ltd., #130-5421 10th Ave., Delta, B.C., V4M 3T9, Canada. Phone: (604) 943-8021.

FUTURE ENHANCEMENTS Two major problems (as well as many minor ones) remain to be fixed in the compiler, if it is to be used in a commercial environment. The first is that character strings must be stored with one character per word, rather than one character per byte. This needs to be fixed so that variables take up less room, and so that PASCAL programs can communicate directly with HP subsystems such as IMAGE.

The second problem has to do with PASCAL's definition of parameters to procedures and SPL's definition of parameters to procedures. SPL is known as a programming language with weak type checking. This gives the programmer more flexibility, but provides more opportunity for making mistakes. It also allows IMAGE parameters to be defined very loosely. For example, a data set can be defined by a character string representing the name of the data set, or by an integer variable with the set number. PASCAL does not allow such flexibility in parameter passing, so that some mechanism will have to be set up to allow procedure calls to subsystems such as IMAGE.

MERGING CONTRIBUTED PASCAL WITH HP/PASCAL

At the Swiss Hewlett-Packard General Systems Users Group (HPSUG) meeting, HP representatives said that HP would provide a supported PASCAL compiler for the HP/3000 some time in the next six to eighteen months. Because of this, PASCAL 2.7-V should be viewed as a temporary measure which will be replaced once HP introduces its compiler. It can be used for staff training and for the development of programs that will be recompiled on the HP unit when it becomes available.

PASCAL/3000 The HP compiler is to be based on an internal HP standard for PASCAL. The HP/1000 PASCAL compiler which was recently introduced is also supposed to follow the internal HP PASCAL standard. Any comments I make about PASCAL for the HP/3000 are taken from how things are done on the HP/1000.

If PASCAL/3000 looks very similar to PASCAL/1000, then we can all look forward to an excellent implementation of the language. PASCAL/1000 provides for all of the features which take advantage of the HP/1000 operating system, but they stay within the confines of PASCAL. Of special importance is the inclusion of a compiler option to permit the compiler to compile STANDARD PASCAL only. By turning this option on, only the PASCAL source that followed the standard would compile.

Storage allocation in PASCAL/1000 is done in a very flexible manner. Two restrictions of the contributed versions of PASCAL are that they do not allow for double word integers, or for sets with greater than 62 elements. PASCAL/1000 permits sets with up to 32767 elements, and only allocates as much storage as needed. Similarly, both single and double word integers may be declared in a way that is very natural for the PASCAL language.

PASCAL/1000 also allows for character strings to be stored as one character per byte, instead of just one character per word. Procedure calls are allowed to external procedures written in either PASCAL or HP/1000 assembler.

Assuming PASCAL/3000 will follow the lead of PASCAL/1000, it should be a very successful compiler. The only problem that I don't know the solution to is how PASCAL/3000 will permit calls to HP subsystems like IMAGE, while working within the confines of PASCAL type checking.

WHAT TO WATCH FOR If you are only interested in using the contributed versions of PASCAL to write portable software, then there should be no problems converting your PASCAL programs to work with PASCAL/3000.

If your programs are designed to work on the HP/3000 only, and if they make any assumptions about the internal storage of PASCAL, or the parameter passing mechanism, then problems could arise. For example, all variables declared INTEGER will take up more storage in PASCAL/3000. Also, the way in which direct access is done to MPE files will change with PASCAL/3000, and file parameters will most likely not be passed in the same manner as they are now.

The users who will be affected the most are those that call SPL or FORTRAN routines using the current PASCAL compilers. After recompiling with the new compiler, the programs may not work.

WHAT TO DO NOW

Since it may be some time until the HP PASCAL/3000 is available, you want to look at PASCAL 2.7-V as a filler. PASCAL 2.7-V will compile almost all features of STANDARD PASCAL and provides a means of learning the language. Included with PASCAL 2.7-V is "A Programmer's Introduction to PASCAL" by Bary Pollack and David Greer. This very brief document reviews the language PASCAL in terms of other languages, such as ALGOL or SPL. Interested people can discover some of the very useful features of PASCAL by reading the document and testing out small PASCAL programs using PASCAL 2.7-V.

Another incentive for having a PASCAL compiler available is that more and more software in the Contributed library is being written in PASCAL. One of the major ones is PASCAL-S, which is a load and go compiler of a complete PASCAL subset. PASCAL-S is being used by several colleges to teach PASCAL (see Appendix I for more details).

Appendix I

MAKING PASCAL PORTABLE—TWO EXAMPLES

I. PASCAL-S

PASCAL-S is a subset of STANDARD PASCAL. The compiler itself is a complete system which compiles the PASCAL program, and if there are no errors, executes the program (interpretively). PASCAL-S itself is written in PASCAL by Wirth.

The major problem in getting PASCAL-S running on the HP/3000 was the difference in the character set on the CDC series of machines and the character set of the HP/3000 (ASCII). In the CDC character set, blank collate after letters; but in the ASCII character set, blank collates before letters. Also, the internal numeric representation of characters differs from the CDC to the HP/3000; and this caused further errors in the compiler.

The lesson to learn from this is that writing portable programs, even in PASCAL, takes some thought and effort. Even when using STANDARD PASCAL, problems can arise. One of the major ones to watch out for is that no assumptions are made about character sets, as they vary widely from machine to machine.

II. PROSE

PROSE is a text formatter published in PASCAL News No. 15. It is written entirely in STANDARD PASCAL and pays particular attention to the character sets of different machines. The solution in PROSE was to store all text internally using the ASCII conventions; and each implementation of PROSE must have a different set of routines to convert from the external character set to the internal one.

PROSE is approximately 3500 lines long, and it is just now being completed on the HP/3000. The main problems encountered in implementing PROSE were finding and eliminating typing mistakes, and understanding the conversion from the external character sets to the internal set. Even though the external and internal character set are the same on the HP/3000, it took some time to find and change the conversion routines accurately. The main problem in this area was a lack of understanding of how the CDC character sets work, since the version of PROSE written in PASCAL News No. 15 was for CDC character sets.

Bibliography

There has been a great deal published about PASCAL over the last few years. The following is a short list of useful documents, especially for those wanting to know more about the language itself and about PASCAL on the HP/3000 in particular.

(1) PASCAL User Manual and Report, Second Edition
Kathleen Jensen and Niklaus Wirth
Springer-Verlag, New York, 1974

(2) Programming in PASCAL
Peter Grogono
Addison-Wesley, Don Mills, 1979

(3) The PASCAL Programming Language
Bob Fraley
Proceedings of the North American Meeting, February 1980

(4) PASCAL-P on the HP/3000
Bob Fraley
Ibid.

(5) A Programmer's Introduction to PASCAL
Bary Pollack and David Greer
Available through Robelle Consulting Ltd.

(6) PASCAL News
c/o Rick Shaw
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342
Subscription rates are $6.00 per year

(7) A Draft Proposal for PASCAL
A. M. Addyman
SIGPLAN Notices Vol. 15 No. 4, April 1980
Association for Computing Machinery,
1133 Avenue of the Americas, New York, NY 10036

(8) PASCAL for the HP/3000
John Earls
JOURNAL-3000 Vol. 1 No. 5

(9) PASCAL/1000 Programmers's Reference Manual
Hewlett-Packard part number 92832-90001

(10) The PASCAL (P) Compiler: Implementation Notes
Order from William Waite
Software Engineering Group
Electrical Engineering Department
University of Colorado
Boulder, Colorado 80309

(11) PASCAL-S: A Subset and its Implementation
Niklaus Wirth
See order information above

INDEXING BRYN MAWR'S FIRST DEAN: AN ESSAY IN DATA ENTRY AND TEXT FORMATTING

Jay Martin Anderson
Bryn Mawr College
Bryn Mawr, PA 19010

BACKGROUND The first dean and second president of Bryn Mawr College, M. Carey Thomas (1857-1935), was a figure of some stature in women's rights at the turn of the century, and especially in the field of women's education. Besides her strong leadership in the founding and early years of the College, Thomas was also a leader in the women's suffrage movement. It is not surprising that her personal and professional papers, especially while dean and president of the College, would prove an important resource to scholars interested in those issues or that period.

Under a grant from the National Historical Publications and Records Commission, the College was able to collect and edit Thomas' papers, and to prepare them for microfilming. This work was directed by Lucy F. West, now archivist for the College. West also provided notes to accompany each reel of microfilm. However, to make the microfilm edition of Thomas' papers more easily accessible and useful to the scholarly community, the archival staff of the library, together with the office of computing services, embarked on a project to create an index of the papers, and to print that index. This index is believed to be the first computer-generated index to the microfilm edition of the papers of a famous American.

The microfilm edition of Thomas' papers, West's reel notes and guide, and the computer-generated index will be published by Research Publications, Inc., Woodbridge, Conn., in 1981.

INTRODUCTION Unlike many data-processing tasks, the M. Carey Thomas Index is characterized by uniqueness: once done, the project will never be repeated. Consequently, the project was designed to minimize programmer's time, even at the expense of less efficient processing tools. Furthermore, the project was undertaken with full awareness of the DP naivete of the archival staff of the library combined with the bibliographic naivete of the computing staff. During the course of the project, many mistakes were made and much interdepartmental education took place. Finally, the project was undertaken on a new, and lightly-loaded Hewlett-Packard HP 3000 computer. The project would have been impossible for lack of computing resources had it begun six months earlier; it would have been impractical if not impossible because of the crowding of those new resources had it begun six months later.

These conditions led to a project designed with the following criteria:

(1) Use to the greatest extent possible, simple, HP-supported utilities and subsystems.

(2) Avoid writing programs; when necessary, use FORTRAN (the language best known to the author).

(3) Use sequential files; don't force the author to learn KSAM or IMAGE when he's yet a novice HP 3000 System Manager.

(4) Use a text-formatting program supplied in the HPGSUG Contributed Library; there was no budget for new software.

(5) Use a letter-quality printing terminal which can deliver printed output as close as possible to typeset, as fast as possible, at the least expense. The Agile A-1 terminal was selected for this device.

To give the reader some feeling for the scope of the project, we cite the following. Nearly 52,000 items of Thomas' correspondence were entered by a staff of over a dozen students and library workers over a three-month period. After initial processing (principally reformatting, sorting, and merging), a raw index was printed for proofreading. The accumulated data amounted to about 20 Mb. The raw data was edited by one worker over a two-month period. After examining many samples, the publisher, the archival staff, and the computing staff, agreed upon an output format. The text-formatting and printing was carried out in seventeen hours over two days, operated by the author's twelve-year-old son. About 1200 $9\frac{1}{2}$ by 14" pages, and nearly a dozen ribbons, were consumed.

The project was divided into six phases: entry, assignment, sort-merge, proofread, edit, and format-print. Because the project was unique, neither documentation nor transportability of code or procedures were deemed important. A brief description of each phase follows.

ENTRY Four different source documents were used: fileslips, on which were recorded data about correspondence in Thomas' office files, usually dated; 5x8 cards, describing the majority of Thomas' personal and professional correspondence, with reference to the scrapbook or letterbook in which the original correspondence was preserved; correspondence referred to only by the number of the

reel of microfilm on which it was recorded; and bibliographic "see" and "see also" references.

V/3000 was used for data entry. Four simple forms were designed, one for each of the four types of documents. Simple field processing was used for data verification, and to simplify entering records for more than one piece of correspondence to or from a given individual.

Four analogous reformat-specifications were designed to transform the captured data simply to 364-byte sequential records.

ASSIGNMENT Simple FORTRAN programs were written to assign microfilm reel numbers to data in the "file" and "card" classes. In the former case, dates in the form ***ddmmmyy*** were converted to Julian form, and a table-lookup procedure used to discover the reel number. In the latter case, the letterbook number (a Roman numeral) was converted to an integer, and a table-lookup performed.

In addition, one of five record types (integers 1-5) was assigned to identify the three kinds of source data and the two kinds of bibliographic references. This step was an afterthought: it could have been accomplished in the entry phase. All of the "assignment" tasks could have been accomplished in the entry phase, had the V/3000 intrinsics been used within a FORTRAN program.

SORT-MERGE Using HP-supported utilities, the individually reformatted batches of data were merged and then sorted. It is perhaps noteworthy that the final sort, of about 52,000 records, on seven fields spanning 131 bytes, was accomplished in 25 cpu-minutes and 70 wall-minutes. The sort was done by carrying both an all-uppercase and a mixed upper- and lowercase version of all personal name fields, and sorting on the uppercase version. The new release of HP's sort-merge would make this extra overhead unecessary. After sorting, the upshifted fields were stripped off, leaving 240-byte records. Finally, in anticipation of the use of the EDITOR, the 52,000 records were divided into four smaller files, and line-numbered.

PROOFREAD A FORTRAN program was written to provide formatted output of the data for proofreading. At this point, the manual procedure of examining each record and marking it for correction followed.

EDIT The corrections marked on the proofreader's list were accomplished with batch edits using the HP EDITOR. Records out of place were moved with the GATHER command. This proved to be cumbersome, and was the only phase where the criteria of using sequential files and HP subsystems proved to be awkward.

The source data also included four small sets of data which were entered and sorted under abbreviated names (BMC = Bryn Mawr College: MCT = M. Carey Thomas: St. = Saint; and US = United States). These data were treated exactly as the main body of data through the proofread phase, and then GATHERed into alphabetical order in this phase.

Format-print. Printing was planned for 2¾" wide, 13" long columns on 14" (legal size) paper, at twelve characters per inch, six lines per inch. An extensive FORTRAN program:

- issued margin information and sent an initial escape sequence to the AGILE terminal;
- read each record of the edited data, and
- issued control breaks for each change of name or change of record-type;
- formatted each name with appropriate punctuation and compressed blanks;
- formatted each data element with appropriate punctuation and compressed blanks;
- issued control-characters for the automatic underlining of the reel number and of the words "see" and "see also" in references;
- issued a top-of-form and reset the AGILE terminal when done.
- copy a subset of the edited data to a new file;

accomplished these steps:

To allow the operator to change ribbons and paper, and to pace the work throughout the two days of printing, the format-print job was run once for each letter of the alphabet. A simple UDC

- process the data with the aforementioned program, placing the output in a second file;
- use the text-formatter FORMAT from the contributed library to control printing.

Final products. At the risk of belaboring the point, the project was unique. Consequently, the only products are these:

- edited data, captured just before printing, and now archived on magnetic tape;
- a single copy of the printed index, now in the hands of the publisher;
- this article, and listings of the programs and procedures which support the six phases of the work.

IN THE BEGINNING

by Marc Covitt — HP/San Diego Division

As part of a project at the San Diego Division to review our present Account Structure on the HP3000 we did a great deal of research to determine just how we got our current structure. Our archivist, Marc Covitt, says he found the following document in a dusty urn buried beneath the raised floor.

In the beginning the movers delivered the HP-3000. And the machine was without complete form and darkness was upon the face of the switch register. And the spirit of the technicians moved about the machine; and the CE said, "Let there be power"; and there was power. And the CE saw that the power was good (or at least within specs) and the machine was operational. And he divided the operational state from the non-operational state. And he called the operational state "working" and he called the non-operational state "down." And it was down and it was working a first day.

And the CE said, "Let there be a System Manager for the working machine and let him partition the system for its users." And there was a System Manager and he collected all the files given to him by the CE in the SYS and SUPPORT accounts from all the files given to him by users. And he called the files given to him by the CE "MPE" and he called the users files "in development." And the system was working a second day.

And the System Manager said, "Let the users be gathered unto many places" and it was so. And the places he called terminals and the connecting together of the terminals he called cabling, and he saw that it was good (or at least neat.) And he said "Let the users bring forth files from their terminals," and the terminals brought forth programs and data and many files; and the data yielded more data after its kind, and these were databases. And it was still working.

And the System Manager said, "Let there be technicians to divide the system from those that it served." And he called the technicians — programmers and programmer/analysts and Data Base Administrators; and he called those that it served — users and user/analysts and also those turkeys. And the system manager started two great programs upon the system so as to cause utilization of the machine. One program he called COBOL and the technicians compiled much and cursed the machine for it gave bad diagnostics. And the other he called QUERY and the users inquired much into their databases and were much confused but still impressed. And the System Manager saw that it still might work.

Soon the users brought forth applications in abundant nature and all manner of files were upon the system. These were PROC files and XEQ files and text of much documentation, and test data and also real data. And users brought forth more users and shared information amongst them. And the programmers saw that it was easy and they too created many files. And the programmers were clever and knew how to multiply and expanded their staff. And there were files on all of the discs and the discs were full. And the System Manager saw that it might not work.

And the hardware was upgraded and made more powerful and more discs were added. And the users required new applications and wanted more accounts. And the technicians received many files but little support from other technicians on high at Corporate. And the users began to create program files and it was difficult to separate these from those of the technicians. And the auditors were much displeased. So the System Manager said "Let us separate the files and the programs. Those that have a priority on the system and are needed we shall call 'Production' and those that are not of this domain we shall call 'Development' and also 'Test' ". And he intended to give Production dominion over the whole machine and with the power to keep cut compilers and other beasts of response time. And he thought this would work.

Thus the environment was established and the account structure had control over all. And the System Manager rested and looked over all he had done. And he was not well because now he knew that it would not work. So he went to his manager and they commissioned a project called the "Revised Account Structure."

CONTRIBUTED LIBRARY

USING "SPECIAL ACCOUNTS"

by Mark Wysoski

Too often, users of the Contributed Software Library either tend to ignore or are completely unaware of the "special accounts" provided on the Release Tape. This is unfortunate because most of the elaborate contributed systems are in this format. The abstracts identify this fact by indicating in which account the contribution is found. However, because the "special acounts" need not follow accepted Library standards, the group structure and naming conventions usually tend to complicate restoration procedures. It is hoped that the selective restore utility LIBREST. INFOBASE. MAINLIB will help to alleviate some of these restoration problems (by allowing restoration of all files related to the contribution); and should be remembered that further modifications of restoration procedures are slated to be made for future releases of the Library. For whatever reasons, the "special accounts" generally tend to be bypassed.

In an attempt to introduce the user to these "lost" contributions, this article will describe one of the "special accounts", LITSCAN, which is a bibliographic retrieval system based on a keyboard approach.

The first step will necessitate building the appropriate account structure to allow restoration of the files. This is most easily accomplished by following the instructions in the INSTALLATION GUIDE which came with Library Tape. Of specific importance are the commands —

```
: NEWACCT LITSCAN, MGR; CAP=IA,
    BA, SF, ND, GL, AL, AM, PM
: STREAM GROUPS
```

The jobstream GROUPS will create the groups DATA and SOURCE under the account LITSCAN, as well as the group structure for all the other Library accounts. One may find it as easy to examine GROUPS. INFOBASE. MAINLIB, and invoke any appropriate commands interactively. This would save the overhead of creating unnecessary group structure.

Once the account structure has been built, the second step involves the actual restoration of the files. There are two major methods which can be used. The first is using the LIBREST utility (while logged on as MGR. MAINLIB, INFOBASE) and restoring the contribution LITSCAN. The second method is using MPE : RESTORE using the fileset @.@.LITSCAN. The files are in compressed format on the tape and will require 4904 sectors to restore in this format. They should be UNPRESSed and, once converted back to Fixed ASCII format, will require 6452 sectors of disc storage.

Next it would be advantageous to review the documentation. This is accomplished by logging on as MGR.LITSCAN and invoking the following commands —

```
: FILE PRINTER;DEV=LP
: RUN FCOPY.PUB.SYS
>FROM=DOC.DATA;TO=*PRINTER;CCTL
>FROM=INDEX.DATA;TO=*PRINTER
>EXIT
```

The documentation, installation procedures, and an index describing each of the files in the account will be printed on the output device designated as LP. Reading this documentation should provide the user with enough information to make LITSCAN operable.

There are a few points which should be clarified. LITSCAN was developed on an HP3000 Series II and III under MITs 1918 and 2011. LITSCAN uses KSAM/3000 and VIEW/3000 (KSAM version). Only the initial input program requires a block mode terminal; all other programs in this system are compatible with any type of terminal. Privileged Mode is used by LITSCAN to extract the USER name (this routine has been in operation at WHITMAN COLLEGE for over one year and has produced no negative effects).

The input program LITMAINT allows the user to enter a 200 character description, three authors, journal information and date for each article that is to have an entry on the data base. Keywords are automatically extracted using the following rules:

Words under 3 characters in length are eliminated

Numbers and special characters are eliminated

Special characters are considered delimiters of words

Words found in the "dictionary of bad words" are eliminated.

The author fields are also extracted and are considered keywords. The "dictionary" is found in the subroutine KEYWORDS and currently has three

words — "THE", "SOME", and "AND". Keywords can also be added through another program, KEYMAINT, which will accept any entry as a keyword.

All entered information can be recalled by keyword using the LITSCAN program. Security has been provided which allows anyone to examine the file, but allows only the original creator of the data to change the data or associated keywords.

The LITSCAN contribution is very flexible as it stands. While fields have been labelled, they are actually quite arbitrary. Keywords are essentially a tabbing system, and allow as many tabs to be affixed to one article as desired. The applications for the LITSCAN system are bounded by the creativity of the user.

A data base has been contributed which contains the articles from past HPGSUG Journals. It allows a good method for testing the LITSCAN system while providing a useful product to the user base.

This article was not meant to give adequate instructions to operate the LITSCAN system; that is the task of the contributed documentation. The purpose of this article was to call to attention, the existence of the "special accounts" on the Library Tape, by highlighting one of them.

Any questions concerning either "special accounts" or the LITSCAN contribution should be directed to:

Mark C. Wysoski
Manager
HPGSUG Contributed Software
Whitman College
Walla Walla, WA 99362
(509) 527-5360

(continued from page 11)

compiler is a modified version of the PASCAL-P4 compiler, which fixes several bugs in the original P4 compiler, as well as providing several extensions to STANDARD PASCAL. Some of the important ones are: built-in procedures to do direct access to MPE files, an "otherwise" label in the CASE statement, and calls to PASCAL procedures which were compiled externally, as well as procedures written in other languages.

The process of compiling PASCAL programs on the HP/3000 is slightly different from the one described above. Instead of assembling pcode into object code, the assembler of the contributed version assembles pcode into SPL, which is then compiled into USL files.

PASCAL Source > PCODE > SPL > USL

Compiler Assembler SPL Compiler

(continued on page 12)

TIPS AND TECHNIQUES

SET UP IDEAS FOR LOCAL USERS GROUPS

Marc Covitt
HP San Diego Division

The HP3000 users in San Diego County (and Baja, California) get together about 2-3 months for technical meetings. San Diego and Baja (Tijuana and Mexicali) has 35 installations of HP3000 equipment. A typical meeting will have 20-25 sites present.

Like many volunteer groups, its organization and effectiveness depend on continuing involvement of a few key individuals. When a prime motivator of the group left, it was hard to get things rolling again. In order to try and maximize user effort and involvement, we tried to establish a structure and some simple procedures to keep things moving smoothly.

At a prior meeting, we presented some ideas to keep things running. The approach was developed by Jim New and Hez Michel, from Beckman Instruments, and Marc Covitt, from HP San Diego Division.

(A) Develop a package for the meeting chairperson.

1. The package includes a checklist for setting up the meeting:
   (a) The best days are Tuesday through Thursday in the second or third week of the month.
   (b) Luncheon meetings are preferred since we usually can get conference space at no charge from a hotel or restaurant.
   (c) An optional activity (typically a site tour or demo) should be planned by the host for after the luncheon. Occasionally, the meeting topics may be be planned to continue into the afternoon.)
   (d) For help in distribution/mailing, the chairperson of the group or HP will help.
2. The package includes sample agendas from prior meetings.
   (a) Usual meeting is
   — Welcome and introduction of all attendees

— HP new products or announcements
— Users group business meeting
— Regional or international users group announcements
— Featured topic
— Luncheon
— Tour and demos optional

(b) Plan enough breaks during meeting so that attendees get good opportunities to talk and exchange ideas.

(c) Announce next meeting site date and host.

3. The package should also include mailing labels for sending announcement and agenda to members. New names and addresses and changes should be provided by chairperson prior to mailing.

(B) Identify duties and responsibilities of chairperson.

1. Serve for one year (first chairman serves only until December 1980).
2. Maintain current name and address list of members.
3. Contact next meeting chairperson approximately one month prior to meeting to see if everything is running smoothly.
4. Serve as primary contact for the group.
5. Keep historical records of all communiques and meeting handouts.
6. Keep schedule of future meeting dates and hosts.
7. Pass meeting host aids on to next meeting host.
8. Help with topic ideas and aid meeting host as needed.

(C) Identify additional responsibilities for the meeting host, including:

1. Selecting topic
2. Contacting and coordinating facility arrangements.
3. Preparing maps and directions to be sent with meeting announcement.
4. Handling R.S.V.P.'s for meals and collecting funds.
5. Providing a meeting recap and keeping user group folder current.

(D) Schedule future meetings.

1. Based on bimonthly meetings, schedule at least two meeting dates for future (next meeting plus the one after that).
(At our March meeting we were able to schedule meetings and meeting hosts all the way through until January '81).

Setting up a local user group meeting does involve some work. We hope that by simply iterating the steps that it takes we will make the job that much easier for future meeting hosts. If other local groups are interested in finding out more about our structure please contact our chairperson, Chuck Stillwell at Union Carbide, (714) 279-4500 or Marc Covitt at Hewlett Packard, San Diego Division, (714) 487-4100, ext. 498.

The ***JOURNAL of the HP General Systems Users Group*** *depends upon contributed articles to fulfill its purpose of disseminating information to members of the association. Current articles are uniformly of high quality and provide excellent information for the membership. However, the Executive Board and the Publications Committee would like to expand these offerings.*

This request is addressed to those readers with ideas to be shared. The Publications Committee encourages all readers to contribute the results of their work. The ***JOURNAL*** *offers an excellent vehicle for publication of current work for the established computer person as well as for the beginning user. Articles of all types are needed. As installations increase, the need to know, to share ideas, and to establish written communications among members of the Users Group increase at an even faster rate.*

Remember, the areas where problems and failures have a high probability of occurring are of interest to a large number of other HP users. Share information on your success and indicate the future direction of your work by writing articles for the ***JOURNAL.***

Fold along line

BUG/ENHANCEMENT POLL

NAME: _____

PHONE: ( ) _____

TELEX: _____

COMPANY: _____

ADDRESS: _____

_____

_____

Please indicate the BUGS that critically affect the operation of your HP3000. Include any number of bugs but indicate only those with greatest impact. The bugs should be identified by their Known Problem Report (Service Request) number as found in the most recent Software Status Bulletin. Please include BUGS with an "open" status only.

_____

_____

_____

_____

_____

_____

_____

Please include one brief ENHANCEMENT Request (hardware/software/service/other) you would like HP to address. Be brief and include a Keyword that classifies your request as specifically as possible, e.g., 7925 Disc/COBOL/CE support.

KEYWORD: _____

REQUEST: _____

_____

_____

_____

_____

_____

_____

_____

Staple here

Fold along solid line

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8 GLEN BURNIE, MD.

POSTAGE WILL BE PAID BY ADDRESSEE

HP General Systems Users Group

Empire Towers
7310 Ritchie Hwy.
Glen Burnie, MD 21061
USA

Executive Offices
HP General Systems Users Group
Empire Towers
7310 Ritchie Highway
Glen Burnie, Maryland 21061
U.S.A.
Tel. (301) 768-4187

ADDRESS CORRECTION REQUESTED

LINFORD HACKMAN
VYDEC, INC.
30 TARN DRIVE
MORRIS PLAINS, NEW JERSEY
07950, USA

Announcing . . . .

HP General Systems Users Group

Spring International Meeting

Orlando, Florida

April 27-May 1, 1981