PLAY IT AGAIN, KSAM

GARY TODOROFF
DATAMASTER COMPUTER SERVICE

INTRODUCTION

Keys have long been a simple method of organizing and accessing data. For instance, a library's card catalog gives multiple key access by author, title and subject; in the phone book, the white pages are keyed alphabtically by name, and the yellow pages, by product or service. Keyed Sequential Access Method (KSAM) uses the same simple approach based on the concept of keys. Perhaps that's why KSAM feels natuaral and familiar to many programmers.

Of course, real world situations do not always lend themselves to simplicity, especially in data processing. Sometimes data structures and their relationships are very complex. But if certain criteria are met, KSAM is the logical choice for a data processing application. Instead of adding to the complexity, KSAM may simplify the approach with easy, direct retrieval of information. I like to think that KSAM stands for Keep Simple Access Methods.

KSAM HISTORY

A brief historical note will help to set the stage as well as clear up some misconceptions. Perhaps initially, in response to the marketing opportunity and demand for compatability with IBM's System/3 minicomputer, KSAM was rather hastily assembled to replace an even earlier effort called RSAM. KSAM was a bit shakey at first.

Problems were often encountered, such as end-of-file pointers not agreeing between MPE and KSAM. Guess-the-actual-number-of--records in the KSAM file became a not-so-popular game played by many programmers. Even more serious, KSAM files could be left in an unpredictable state after system failures, with nobody the wiser. Folks accustomed to the rock-solid dependability of ISAM (Indexed Sequential Access Method) on the IBM systems of the 1970's encountered some rude suprises with the early KSAM.

However, except for one notorious "fix" to KSAM (more later), conditions have improved greatly. The original VIEW/3000 (now VPLUS) used KSAM for its form files, providing HP good incentive for a major overhaul of KSAM back in MPE release 1918. KSAMUTIL, the HP utility used to build, study, and fix KSAM files also received new features to help solve some of the old problems. Most notable was the "crash flag" (which will let KSAM automatically recover and correct a file damaged by a system failure.) File problems are also quickly fixed with the KEYINFO command in KSAMUTIL which will resolve any key-file problems. File locking was another early improvement to KSAM, enhanced even more when dynamic locking became available to RPG with the "LOCK" and "UNLOCK" operations in a more recent release.

Overall, the improvements have been needed and appreciated. But one "feature" which overwrites KSAM records must be mercilessly exposed! During all those past improvements, KSAM was supposedly modified to be more consistent with IBM ISAM, specifically update operations to a KSAM file when no record was retrieved prior to a file update. For example, consider a batch file which is used to update a KSAM master file. If the batch file is empty, a record in the KSAM file may be overlayed with blanks. Apparently, unless special precautions are taken, the KSAM file is rudely updated while the logical file pointer is still relaxing at the first record awaiting where to go next. With no input batch records and no place better to go, WHAM, there go your empty input batch record buffers, splattering

whatever used to be in the first record of the KSAM file.

Especially with RPG programs, first records can be overwritten. The condition has existed for three years now, causing incredible amounts of confusion, file repairs, reprogramming, and possibly even causing undetected file errors produced by programs which use to run predictably and flawlessly. (Figure 1 shows a sample RPG program in which the problem could occur and the fix required). Known Problem Reports (KPRs) have been submitted to HP concerning the problem many times. So far, the only acknowledgement is a warning on compiler listings of programs which could encounter this situation. Until HP fixes this problem, be forewarned!

(Late news from HP indicates that the above problem may be solved by allowing "Update-Protect-Checking" (UPC) in RPG version 5.08 to be relased with MPE V, S-MIT. UPC is enabled by placing a "U" in column 28 of the RPG Header specification. Using this new feature, indicators would not need to be set on in the calculations as shown in the RPG program of Figure 1.)

KSAM STRENGTHS

Disclaimers now aside, let's proceed to the practical use of KSAM by outlining its strengths for data base functions.

- Both keyed access and chronological access may be used with KSAM files. While the key-file maintains the index, records are added sequentially, resulting in a sometimes useful chronological record order.

- KSAM files may be separated into file extents, thereby not claiming disproportionate amounts of disc space when not full of data. (IMAGE allocates all space immediately and even backs up the blank records to tape.)

- KSAM files may be filled right up to End-of-File without sacrificing performance, again resulting in more effecient disc usage.

- KSAM files do not require special utilities for loading and unloading data. Simple FCOPY commands may be used after a KSAM file is built. In fact, FCOPY itself will build a KSAM file of the same characteristics as the original KSAM file through a very simple use of parenthesis in the FCOPY command. For example:

```
FCOPY FROM=KSAMFILE;TO=(DATAFILE,KEYFILE)
```

FCOPY will create a new file called "DATAFILE" which will be identical to and have the same key-file structure as the original file, "KSAMFILE".

- KSAM key fields may be updated directly.

- Deleted data is retreivable. Since only the key is removed from the key-file, the data file still contains the deleted records which may be retreived later if necessary. Again, using the FCOPY example, deleted records may be recovered using:

```
FCOPY FROM=KSAMFILE;TO=NEWFILE;NEW;NOKSAM
```

\column 2

The first two bytes of the deleted data record were overlayed when the record was deleted, but otherwise the record is still intact.

- Keys may be defined as unique or with duplicate key values allowed. One recent feature to possibly increase performance with duplicates is the "RDUP" parameter of the KSAMUTIL BUILD command. RDUP allows for duplicate keys without maintaining the chronoligical order in which duplicate key records were added to the key-file.

- Standard MPE file handling may be used in most cases with KSAM files. For example, file equations allow a KSAM file to be opened with ";ACC=OUT", thereby easily setting the file pointer to the beginning of the file, deleting any previous data. Even the EDITOR may be used to make occaisional or global changes to a KSAM file, for example:

```
:EDITOR
/TEXT KSAMFILE,UNN
```

```
/CHANGEQ "PARTNO  " TO "PART-NUM" IN ALL
/KEEP $NEWPASS,UNN
/EXIT
:FCOPY FROM=$OLDPASS;TO=KSAMFILE
```

- Perhaps one of KSAM's biggest advantages is to allow a file to be processed sequentially by key. Especially if listings by key are often required either on paper or the terminal, this provides a real advantage over the multi-step process with IMAGE, which may involve extracting data from a data set, sorting it and finally listing it. Some users have stored just key information in KSAM files, with IMAGE used for storing the actual data, providing an external sequential by key access to IMAGE data bases.

- An easy method of jumping into a KSAM file is allowed with partial key or generic key access. From that point in the file, records may then be processed sequentially by key. Figure 2 provides a sample of how to do this with RPG.

KSAM DISADVANTAGES

Before mentioning some special aspects of KSAM, a few of the disadvantages should be mentioned, too.

- HP does not provide a QUERY-like language for accessing KSAM files.

- Except for standard file lock words, KSAM does not allow for file security at the record and/or field level.

- Keys must be contiguous. Multiple keys may overlap, but cannot start in the same position.

- Without careful locking strategies, concurrent record access may cause problems.

- Interrelations in data sets are not automatically handled by KSAM , but must be done within the application software.

- KSAM opens an Extra Data Segment for each file and user, which can increase resource usage and require special techniques for file sharing.

Of course, one usually finds a way around most disadvantages, or at least some way to excuse them. I won't excuse any, but will mention some of the software and techniques available to overcome some of KSAM's limitations.

KSAM UTILITY PROGRAMS

KSMQUERY from the Contributed Library provides a "quick and dirty" access to KSAM files. Primarily intended as a file debugging tool, KSMQUERY does open up files very simply for inspection and extracts records easily by key values of equal to, less than or greater than. If for example, invoice number is a key, and you want to see all records since invoice number 1000, the KSMQUERY syntax looks like this:

```
>FILE INVOICES      (specify file name)
>RECS 50            (display up to 50 records)
>SEARCH GT          (show records with keys greater than)
>=1000              (start search at key value, 1000)
```

KSMQUERY will then display records to terminal or line printer.

KSMQUERY provides an additional capability which is unique on the HP3000. By using the "DUMP HEX" command, packed fields are very easily read with the four bit "halves" of each byte displayed in decimal form below each character. Until I either lose two fingers or gain six, octal and hexidecimal will never feel comfortable, so it's a real pleasure with KSMQUERY to read packed fields directly. (See example in Figure 3) In fact, a couple of times I've even used FCOPY to put a file subset

into a quickly built KSAM file just to use the HEX display feature, a very handy way to read packed fields.

Another useful program from the Contributed Library is KSAMRBLD. KSAM files can be modified by answering interactive prompts for blocking factor, file limit, and number of extents. It will even pick out an effecient blocking factor for you, then rebuild the file with your new sizes.

KSAM DEFAULT FEATURES

One of the advantages of KSAM is the way it defaults to workable file structures if you don't provide the parameters. Obscure areas such as key blocking factors will be taken care of automatically. In fact, the best approach is often not to think too hard about all the various options. For example, the FIRSTREC option of the KSAMUTIL BUILD command allows you to override the normal first record equals zero default. (To begin counting at "zero" rather than "one" never ceases to amaze me; people have counted starting with the number one since recorded history. Then some computer engineer decided to start counting with zero. Future archeologists will no doubt puzzle over this strange practice.) Nevertheless, use the KSAM default of zero for consistency and everything works fine.

MULTIPLE KEY USE

KSAM allows for an amazing sixteen possible keys per record--most of my files don't have that many fields. I have always been curious to try out so much horsepower, just to see what happens. Actually, two keys is the most I've ever used, which keeps me well within the usual warnings to avoid using much more than three keys per file. Of course, more keys could be used, especially for files that are primarily for inquiry, with updates being done only rarely or where speed and disc I/O are not important, such as adding records in batch mode to an archival file that is updated monthly. Adding records with multiple keys can cause a large amount of reshuffling the key-file, so consider the available resources carefully.

One of those occaisonal two-key files I have used provides an interesting example of the KSAM key features. A file of job records needed to be accessed both by part-number/job-number(a key made up of the two fields) and by part-number only, but in date order. The original sequential file was sorted by date within part-number before copying it into a KSAM file. The job-number was not used as a sort field, even though it will become part of the KSAM key. But note that both keys involved the part-number. Keys may overlap but not start in the same position. To solve that, a blank character was always placed to the left of the part-number itself so that one of the keys could start in that position. The record layout was something like this:

```
 column
position..  3   4       8                 32
        /BLANK/PART-NO/JOB-NO/...data.../DATE/
        ^^^^^^^^^^^^^^
          key #1 ^
                 ^^^^^^^^^^^^^^
                      key#2 ^
```

The various steps to create this file are as follows:

```
:RUN SORT.PUB.SYS
INPUT JOBMPE
OUTPUT JOBSORT
KEY 3,4          (the part-no field)
KEY 32,6         (the date field in YYMMDD format)
END

:RUN KSAMUTIL.PUB.SYS
BUILD DJOB;REC=-80,16,F,ASCII;DISC=150000,15;&
KEYFILE=KJOB;KEY=B,3,5,,DUP;KEY=B,4,8,,DUP
EXIT

:FCOPY FROM=JOBSORT;TO=DJOB
```

Access by key #2 is rather straight-forward, since records can be processed by part-number and job-number keyed sequence. Key #1 provides a more interesting access. First, duplicate keys are used. Since the DUP parameter (as opposed to RDUP) was used in the KSAMUTIL BUILD command, the chronological order of the key file was maintained as the records were added to the KSAM file. Therefore whenever records are accessed by key #1, they are always in date order, (since the original MPE file used date as a sort field) even though the date field is not part of any key. Also note that by starting key #1 with a blank character one byte to the left of the part-number, we overcame the limitation which does not allow KSAM multiple keys to start in the same position. Note too that key #1 starts in postion three of the record, leaving the first two bytes empty since KSAM uses that word as the delete flag.

Still on the same file example, an interesting mistake occurred when loading this file which contained around 150,000 records. Due to a typo, the sort was not done by the part-number key field as planned. Thus the "sorted" file was in random order from the standpoint of the key. The file took over six hours to load. After the problem was discovered, the file was resorted correctly, and FCOPY accomplished the load in about 50 minutes. (The computer was a very lightly loaded Series 44) Draw your own conclusions on how to order files before batch adds or updates to KSAM files.

SETLL COMMAND FOR KSAM FILES IN RPG

One final example using the above file involves a simple coding technique in RPG demonstrating how to sequentially process the file beginning with a selected part-number. The RPG calculation operation, SETLL (Set Lower Limit) is used with the KSAM file defined as Input Demand. Records are read and output for as long as the part-number matches the one selected. (See Figure 2)

KSAM FILE SHARING AND LOCKING

Straying farther from the Keep Simple Access Methods philosophy, let's discuss file sharing and locking. Again the examples will be given in RPG. One of KSAM's disadvantages concerns the use of Extra Data Segments (XDS) for each file that is opened by each user. Since a global control for data is not available, some special techniques need to be used to insure data integrity.

In a shared file environment, one file accessor has a personal "snap-shot" of data and key-structure stored in one XDS per user. The danger is that a record may be retrieved and changed while a second user has completed the same, resulting in a modified key-structure which no longer corresponds to the XDS information of our first user. When the first user then updates the file, the wrong record could be updated, since record pointers may no longer be valid. (To explain fully would require a lengthly discussion of the KSAM "B-tree" used for indexing files. References are given in the bibliography, since internal KSAM structures are beyond the scope of this article.)

Locking not only needs to be employed, but must also be done with the right technique to insure that the current and accurate copy of the key-file is in the XDS as the time the file update is done. Primarily, this requires that the file is read once to get a data record, then read once more to re-position the file pointer in case any changes were made to the key-file by another user. By locking around both operations, data will be updated properly. Optionally, the file could simply be locked for the entire inquiry and update operation. The file would be unavailable to other users while one user has a record displayed on the terminal. This technique has drawbacks and could cause definite problems amongst users unless, of course, they all take their coffee breaks together.

If extensive file sharing will be done with KSAM, then please read HP's Communicator Number 21. Otherwise the simple solution is to just avoid file situations that require KSAM file locking and sharing.

FILE STRUCTURE AND BLOCKING

Blocking factors for KSAM data files may be approached in the same way as MPE files, that is, try to use disc space effeciently by blocking on sector boundaries. However, disc utilization is not the only factor, especially if the KSAM file will most often be accessed randomly. A large blocking factor for the sake of disc usage could create a waste of memory by requiring a large XDS to contain the data block. In a large file, the chances are remote that the next

randomly accessed record will be in the same data block already in memory; large blocking factores will not improve disc I/O. Keep data blocking factors small for files that are usually accessed randomly.

If, however, the file will be processed sequentially most often, then consider the average number of records that make up a "unit" of data, for example the number of entries per customer invoice. If that number is 15, then a blocking factor of approximately that size would be appropriate to avoid extra disc I/O if displaying an invoice for that customer. As usual, the trade-offs are between disc and memory usage, and should be judged by the type of file being used.

Blocking factors also apply to KSAM key-files. Once again, I must defer to another article which explains in detail how significant performance improvements may be obtained through the use of effecient blocking factors for key-files. I strongly reccommend that you obtain Jorge Guerrero's article on KSAM Design Guidelines. His paper provides one of the clearest explanations available of the B-tree structure, and explains how to use blocking factors to keep B-tree levels to a minimum, thereby decreasing disc I/O very significantly. The tables included in Guerrero's article go way beyond anything provided in the KSAM manual, yet are simple to use for optimizing key-file blocking factors. If you are at all concerned with KSAM file performance, those key-blocking tables are a must. (See Bibliography)

For further information, Appendix B of the KSAM manual is well worth reading. File structures are covered in depth.

CONCLUSION

My favorite part in the KSAM manual is the first sentence on page B-1, which provides a good point for concluding:

> "KSAM files can be used efficiently without any knowledge of how the files are structured or how file blocking and size is determined."

In other words, you don't have to be an expert to use KSAM on the HP3000. The simple approach of using keys to retrieve data provides information in a way as easy to understand as the yellow pages. KSAM files are easy to create and maintain. They are also relatively effecient in their use of disc space. Various utility routines let you manipulate KSAM files, and the ease of defining and reading KSAM files with RPG gives an especially high-level approach to programming. If the best approach can be defined as the simplest approach, then don't allow things like data structures to become complicated in the first place. For keeping simple access methods, KSAM is a good place to start.

FIGURE 1. "Work around potential corruption of first record in KSAM update file by conditioning the update with an indicator set on in calculation specifications."

```
     H
     H* A BATCH FILE UPDATES A KSAM FILE.  BE SURE TO SETON
     H* INDICATOR 90 IN CALCULATIONS TO PREVENT CORRUPTING THE
     H* FIRST RECORD OF THE KSAM FILE IF THE BATCH FILE IS EMPTY
     H*

     F* BATCH INPUT FILE
     FBATCH   IP  F  80  80           DISC
     F* KSAM UPDATE-ADD FILE (COL-15="U" AND COL-66="A")
     FMASTER  UC  F 128 128R 6AI     3 DISC

     IBATCH   NS  01
     I                                        1   6 INKEY L1
     I*... etc
     IMASTER  NS  02
     I                                        3   8 KSKEY
     I*... etc

     C* SETON INDICATOR FIRST TIME CALCULATIONS ARE DONE
     C*
     C  N90                SETON                      90
     C*
     C* CHAIN TO MASTER FILE TO BE UPDATED
     C*
     CL1         INKEY     CHAINMASTER                66
     C*... etc

     O* UPDATE MASTER FILE AT LEVEL BREAK L1 WHEN INKEY CHANGES
     OMASTER  T       N66 L1 90
     O*...output fields, etc
```

Note that the output is conditioned by indicator 90 which will only be on if there were records in the BATCH file. If the output was not conditioned by 90 and the BATCH file was empty, the first record in the KSAM file would be overlaid with blank field buffers, since indicator 66 is not on and the L1 level indicator is set on along with the last record LR indicator.

FIGURE 2. "The RPG Set Lower Limit (SETLL) calculation allows a simple way to process a KSAM file sequentially within a defined limit."

```
     H
     H* THIS PROGRAM READS A KSAM FILE SEQUENTIALLY WITHIN
     H* LIMITS USING THE "SETLL" CALCULATION.
     H*

     F* KSAM INPUT-DEMAND FILE (COL-15="I" AND COL-16="D")
     F*
     FJOBFILE ID  F  80  80L 5AI     3 DISC
     F*                   ^ ^        ^ KEY STARTS COLUMN THREE
     F*                   ^ ^ KEY IS FIVE BYTES LONG
     F*                   ^ PROCESS BY LIMIT
     F*
     F* OUTPUT FILE DEFINED ON LINE PRINTER
     F*
     FOUTFILE O   F 132 132            LP

     IJOBFILE NS  01
     I                                        3   7 PART#
     I                                        3   5 PART3
     I*... etc

     C* ACCESS TO KSAM IS "GENERIC" BY FIRST THREE BYTES
     C* OF THE PART NUMBER.  SET UP A PARTIAL KEY TO
     C* USE AS KSAM FILE POINTER
     C*
     C                    MOVE "123  "   KEY05   5
     C           KEY05    SETLLJOBFILE
     C*
     C* DO READ LOOP UNTIL FIRST THREE BYTES OF KEY DO NOT MATCH
     C*
     C           RLOOP    TAG
     C                    READ JOBFILE                  60 E-O-F
     C  N60      PART3    COMP "123"              60     >
     C  N60               EXCPT
     C  N60               GOTO RLOOP
     C*... etc

     O* OUTPUT IS TO PRINTER BASED ON "EXCPT" WHILE KEY IS
     O* EQUAL TO "123" IN FIRST THREE POSITIONS.
     O*
     OOUTFILE E       N60 01
     O*...output fields, etc
```

FIGURE 3. "KSMQUERY allows packed decimal fields to be read easily by displaying each half of the byte in decimal format."

Type the HEX command in KSMQUERY before displaying a record. The record will display on the terminal 40 bytes at a time with column indicators.

In the example below, "^^^^" denotes a birthdate field in bytes 35 to 38 of the record. The field is stored in packed decimal format. By reading the bottom two rows diaganolly, starting with the lower zero, the date can be read as "071256". The "q%l" on the first row is "garbage" that comes from interpreting the field as ASCII.

```
                1...5....0....5....0....5....0....5....0....5
1 ...   40  00001805SMITH           HENRY   KM.q%10
            2333333335445422222222224445522244072642
            0000018053D948000000000085E29000BD015CF0
                                                ^^^^
```

BIBLIOGRAPHY

HP COMMUNICATOR, NUMBER 21, MPE 1918, "TIPS ON INSURING KSAM FILE INTEGRITY", P. 13

FOLKINS, DALE, HEWLETT PACKARD, "KSAM - IT'S ALIVE AND WELL" CONFERENCE PROCEEDINGS, P. L-9

GUERRERO, JORGE, HEWLETT PACKARD, "KSAM DESIGN GUIDELINES FOR OPTIMIZATION", IUG COPENHAGEN PROCEEDINGS, 1982

KAMINSKI, STEVE, "KSAM VS. IMAGE", JOURNAL OF THE HP GENERAL SYSTEM USERS GROUP, VOL 1 NO 6, MARCH/APRIL 1978, PP. 16-18

As General Manager of Datamaster Computer Service, Gary Todoroff has been serving a diversified group of clients since 1975. Along with consulting in systems management and programming, he also directs five other programmers in applications on several HP3000 computers and a few micro and IBM computers. Specialties include IBM to HP conversions, programmer productivity tools, and applicatons in health-care, wholesale distribution, direct mail, and fund-raising. Most recently, he has been especially involved with designing and marketing ORBIX Control Language, an application supervisor system which also allows IBM System/34/36 software to run directly on the HP3000.

Based in Eureka, California between San Francisco and Portland, Gary is involved with both the Bay Area (BAYRUG) and Northwest (NOWRUG) Regional User Groups. He started HUMBUG, Humboldt County's own user group and is a supporter of RPGSIG, which he hopes will encourage use of RPG and other high-level language use on the HP3000. Gary may be reached in Eureka at 707/445-8425.
