A New Tool for Fast Keyed Access (Sometimes)

Jim Kramer, HP St. Louis

I. Introduction

This paper discusses a method of keyed sequential access to a file which should for certain uses provide performance advantages over KSAM/3000. It also discusses a set of procedures which implement the method. In addition to keyed access these procedures can be used for sequential reading and writing of a non-keyed file; because they use NOBUF I/O, they can provide considerable CPU savings over more usual methods of sequential access.

The set of procedures -- referred to in this paper as Quick-key -- are available from the contributed library, along with full documentation.

II. The Method

In doing keyed sequential reading of a file, it seems obvious that there should be some advantage to be gained in sorting the file first. There is indeed an advantage. For an unsorted file, knowing the location of a particular record of the file does not provide any help in finding any other record (except to know that it must be different). For a sorted file, however, knowing the location of a record with a particular key tells you immediately that all records with smaller keys precede the known record, and all records with larger keys follow it. If the keys for the first record of each block are known then it is possible to know immediately without looking at the file which block a particular record is in. The record can then be obtained with a single disc access.

This is the method the Quick-key procedures use. They maintain in memory an index containing keys of the initial records of the blocks of the file. This index can either be pre-built (by a Quick-key program which would be run after the file is sorted) or built dynamically by Quick-key. When Quick-key does not have a complete index of the file, it uses a combination of binary search and interpolation to find the desired record. As it does so it obtains more information for its index.

Because the file must be sorted, the modes of access are limited. Reading and updating (without changing the key) are possible; adding and deleting are not.

III. Capabilities

The Quick-key package provides the following capabilities:

1. Read access or read and write access to a file.
2. Exclusive or shared access to a file.
3. For a keyed file:
    a. Reading and updating (without changing the key) of any record of the file.
    b. Random access by sort key, including generic and approximate searches.
    c. Keyed sequential access (accessing the record with the next larger or next smaller key).
    d. Handling of duplicate keys.
4. For an unkeyed file:
    a. Reading or updating of any record.
    b. Appending of records to the end of the file.
    c. Random access by record number.
    d. Forward or backward sequential access.

IV. Usage

Since Quick-key is an unsupported utility, it probably should not be considered for use unless either

1. it provides a desired capability not available elsewhere, or
2. it provides significant performance advantages.

The one significant capability Quick-key provides is the ability to provide keyed access to a sorted file. This capability has already been made use of in a text editor which operates directly on a source file. (A numbered source file is, after all, a file which is sorted by line number). The desire to have such an editor was, in fact, the reason that Quick-key was created.

Significant performance advantages are most likely to be found in the following two areas:

1. Multi-user shared read access to a keyed sequential file, and

2. Sequential reading or writing of a non-keyed file.

Quick-key may provide a large savings in disc accesses over KSAM for keyed sequential access. Quick-key can access a record by key with at most a single disc access. KSAM, on the other hand, may require as many disc accesses as the number of levels of the key file B-tree plus one for the data block. (KSAM may require fewer accesses if the necessary records of the B-tree are already in its buffers).

Whether there will be a memory savings is a complex issue, depending on many factors. In order for Quick-key to be able to access a record with a single disc access, it must keep one key per data block in memory. KSAM, to do the same thing, would have to keep one key per record (the entire key file) in its buffers. Therefore Quick-key will have a memory advantage proportional to the blocking factor of the data file. KSAM, of course, does not need to have the entire key file in memory, but to the extent which it does not it will pay a penalty in disc accesses.

Both Quick-key and KSAM provide a buffer for an entire data block, so the size of the data block affects both in the same way. If there is significant sequential access by key, a large data block helps Quick-key save accesses to the data file -- the next record is probably already in the buffer, and no accesses whatever are required. For KSAM, unless the data file is sorted, an access will probably be required. However if access is largely random, large blocks can waste space for both Quick-key and KSAM -- the next record desired is probably not in the buffer. Of course If access is entirely random by key, an IMAGE master should be considered for use rather than either KSAM or Quick-key.

For sequential access to a non-keyed file, things are simpler. Because Quick-key uses NOBUF I/O and reads or writes an entire data block at one time, the number of calls to the file system can be greatly reduced, depending on the blocking factor of the file. CPU times can be reduced by as much as an order of magnitude as a result.

One other factor that should be considered is that Quick-key's buffering is entirely in the stack, whereas KSAM's is in an extra data segment. Thus a process which requires a large stack for other uses may not be able to use Quick-key. Also, if access to the keyed file is infrequent, Quick-keys buffers could become a useless burden to the stack, whereas KSAM's extra data segment could be swapped out of memory. Another consideration with Quick-key is that

the stack is frozen by MPE while awaiting completion of an I/O request. If such frozen stacks are a significant part of available memory, and if memory is in short supply for the currently executing processes, the memory manager's task can be made significantly more difficult by the frozen stacks.

V. Conclusion

The Quick-key package can provide performance improvements over conventional methods of accessing keyed and sequential files. However the user should be aware of the risks of using software not supported by Hewlett Packard, and satisfy himself that the performance benefits are worth the risk involved.