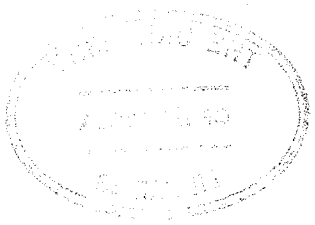
A Primary Key Retrieval System for the PDP-11

by

Gregory A. Matthew

A Thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba, 1980



A PRIMARY KEY RETRIEVAL SYSTEM FOR THE PDP-11

BY

GREGORY ARTHUR MATTHEW

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

ACKNOWLEDGEMENTS

The author would like to express thanks to his advisor, C.R. Zarnke, for his patience and for his very many helpful suggestions about the content and wording of this thesis.

The author also wishes to thank Mrs. Lyn Burkowski for her ability to stretch one week into one month.

The author would like to thank Peter Buhr for his help in the preparation of this manuscript.

Finally, the author wishes to thank his parents without whose constant nagging this thesis would never have been finished.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter I

INTRODUCTION

This thesis describes the architecture of a primary key retrieval system for the Digital PDP-11 minicomputer. A primary key retrieval system is a data management system that identifies data records by a single or primary key. The actual primary key is a field within a data record that can be used to distinguish that particular data record from all other possible data records.

The primary key retrieval system must be capable of locating, by its key field, a particular data record within the data structure and returning or retrieving that data record. The primary key retrieval system must also be capable of adding data records to the data structure in such a way that the new data record fits easily and naturally into the data structure. The primary key retrieval system must have the capability of data record deletion. The final requirement of a primary key retrieval system is the ability to modify data records within the data structure. A primary key retrieval system may, optionally, support sequential data processing during which records are accessed in order by key.

To summarize, a primary key retrieval system identifies each data record by a unique key field. The system must be capable of retrieving, adding, deleting, and updating the data records in the data structure. Optionally, the primary key retrieval system may have the capability of processing the data records sequentially.

The keyed/sequential access method, KSAM, record management system described in this thesis is a primary key retrieval system. The KSAM system identifies each data record by a single key field and is capable of performing all of the operational requirements of a primary key retrieval system. As the name of the system suggests, the KSAM record management system supports sequential data processing as well. The KSAM record management system provides a complete system for the creation and maintenance of data within operating system files based on primary key retrieval.

Chapter II of this thesis contains a description of the objectives sought in the design of the KSAM record management system. These are the objectives considered in addition to the basic requirements of a primary key retrieval system. This chapter sets out the specific design descisions that were incorporated into the KSAM record management system. Some consideration is given towards the rationale for each of these design descisions.

Chapter III contains a description of the entire KSAM data structure including the overall data structure as well as a detailed description of its various components. Brief explanations are offered to demonstrate how the data structure supports the descisions and objectives expressed in the previous chapter.

Chapter IV contains descriptions of the algorithms used to locate data records within the KSAM data structure. Algorithms are given for both methods of record location: location by key and location by sequential position.

Chapter V contains descriptions of the algorithms used to modify the contents of the KSAM file. Three algorithms are described: insertion, deletion, and data record modification.

Chapter VI is a detailed explanation of the processing required to split an information block within the KSAM data structure. Since there are several different types of information blocks different algorithms are used for splitting each.

In addition to the algorithms for the splitting of individual information blocks this chapter contains a section devoted to the processing involved in a multiple block split.

Chapter VII contains desciptions of the various routines that comprise the KSAM record management system. A detailed description of each routine, its arguments, and its effects on the data structure forms the largest part of this chapter. These descriptions form a basic User's Guide for the KSAM record management system. Chapter VII also contains a section listing the codes that can be returned by the KSAM routines. The circumstances that cause each particular code to be generated are explained.

Chapter VIII is a discussion of how well the KSAM file system meets the objectives set forth in the second chapter.

Chapter II

OBJECTIVES OF THE KSAM RECORD MANAGEMENT SYSTEM

In the implementation of the primary key retrieval system, KSAM, seven secondary objectives were selected in addition to the basic objectives of all primary key retrieval systems. As stated in the first chapter, a primary key retrieval system must be capable of identifying data records by a key field and be capable of adding, deleting, retrieving, and modifying these data records. The KSAM record management system attempts to achieve all of the additional objectives without sacrificing the efficient implementation of the basic objectives.

The KSAM record management system had to be able to keep track of more than one data record within the data structure. The ability to quickly locate more than one data record at a time facilitates many data processing tasks. For example, suppose that one of several data records must be selected on the basis of one of the fields in the records other than the key field. In this case the ability to locate each data record independently would be a definite asset.

During the course of updating the KSAM file, data records would be deleted from the KSAM data structure. When a data record is deleted from the file the KSAM record management system had to be capable of reusing the space occupied by the deleted record. This would keep the ratio of unused space to useful information at a minimum. The lower this ratio the better the processing of the file structure. The total space occupied by the file and the time required to process the file would both be lessened. The recovery of unused space would also facilitate later data record insertions.

As the KSAM file is updated, data records would be inserted into the KSAM data structure. Eventually the KSAM file would require expansion to allow for further insert operations. The KSAM record management system should be able to expand the KSAM file with little effort and with a minimal amount of disruption to the current KSAM data structure.

Many indexed file systems operate inefficiently unless the initial loading of the file is done with the records sorted into descending order by key. In these systems if records are loaded in ascending order by key then each new record has a higher key than the last record in the file and consequently forces a new block to be created. People tend to think in terms of ascending order; the name ADAMS comes before the name BROWN and the number 7 before the number 9.

The KSAM file system had to be capable of loading the KSAM data structure with records sorted, by key, into ascending order without an excessive amount of work.

The KSAM record management system had to be capable of processing the KSAM file both by key and by sequence with a minimum amount of work. The combination of keyed and sequential access provides additional flexibility. The key can be used to locate a starting point anywhere within the KSAM file. A series of related data records can then be retrieved sequentially. Most information systems require both of these types of access. For example, most information systems are updated by key but reported on sequentially.

Input and output operations are the slowest and most expensive operations on any computer system. This is particularly true on a minicomputer like the PDP-11. Minicomputer input and output operations are expensive, in terms of processing time, because of the lack of channels to perform the actual input and output operations. The KSAM record management system had to make an effort to minimize the number of input and output operations required.

The KSAM record management system had to provide data integrity. Data integrity means that the data returned to the user is the same data that the user put into the KSAM file. Data integrity also means that the data within the KSAM file is safe. It will not be lost due to system or power failures.

To summarize, the KSAM file system is designed to meet seven objectives beyond the basic ones of any primary key retrieval system. The KSAM record management system is designed:

1. to keep track of more than one data record at a time;

2. to reuse free space created by the delete operation;

3. to allow for easy consistent file expansion;

4. to allow the KSAM file to be loaded in ascending order;

5. to support both indexed and sequential file access;

6. to minimize the number of input and output operations required; and

7. to provide data integrity.

Chapter III

THE STRUCTURE OF THE KSAM FILE

3.1 GENERAL DESCRIPTION OF STRUCTURE

A KSAM file is composed of a series of interrelated memory blocks. These memory blocks may be one of three types: root, index, or data. The blocks are linked together to form a tree structure with the data blocks at the bottom of the structure, the index blocks in the middle, and the root block on the top.

Each type of memory block contains individual information records. These information records may contain either user data or KSAM index information as well as control information for use by the KSAM system.

The data blocks, at the lowest level of the tree, contain information records containing user data; these will be called data records. The data records contain the KSAM user's information. The user has direct access to the data records and may add, delete, change, and retrieve data records in the KSAM file.

The index blocks and the root block contain index records. The index records are used to provide a link from one information block in the tree to another information

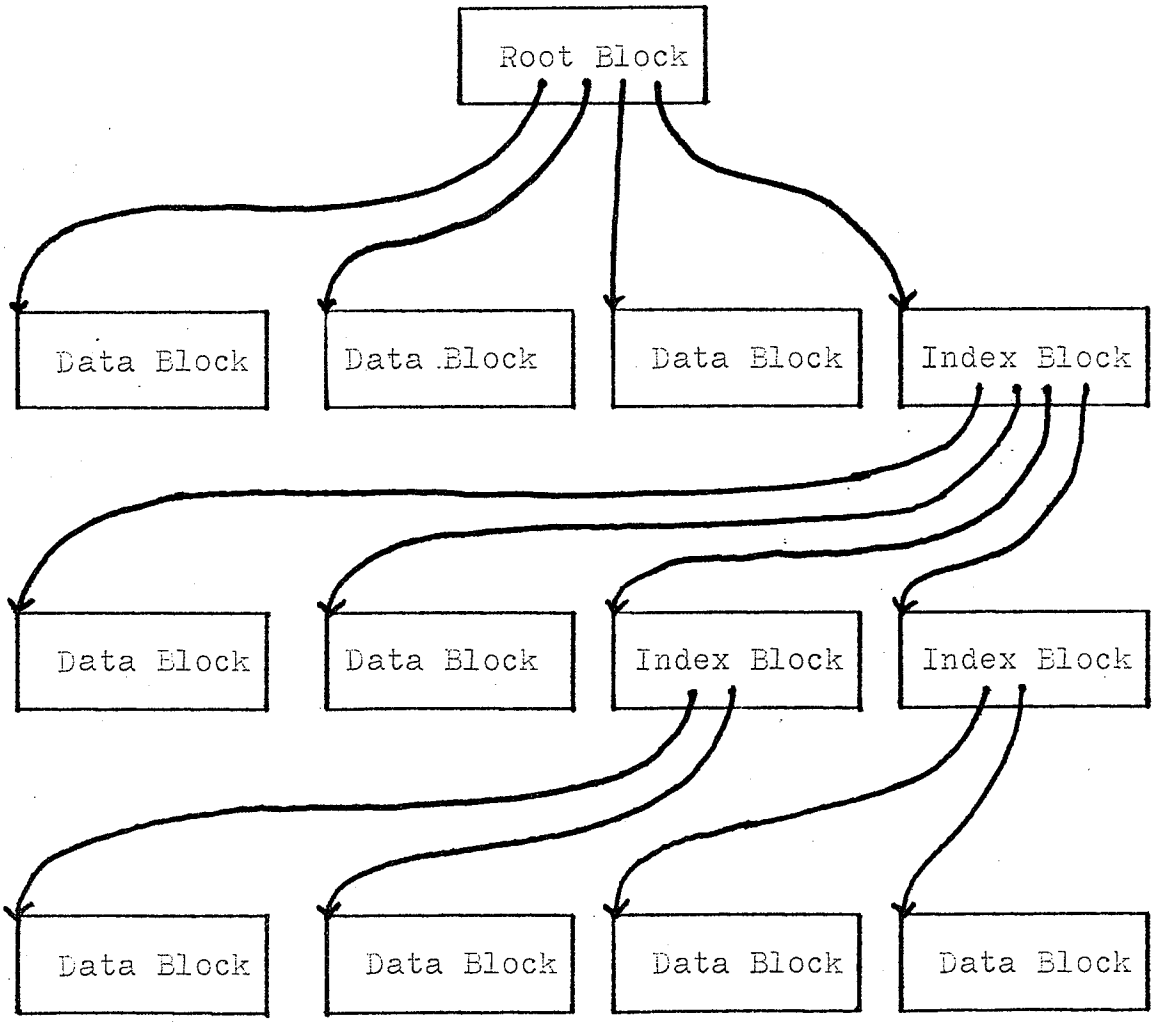


Figure 1: The KSAM Data Structure

block on the next lowest level of the tree. The index records are not accessable to the user. Index records are added and deleted by KSAM to reflect changes made to the tree structure.

3.2 THE KSAM RECORD STRUCTURES

3.2.1 The Data Record

A KSAM data record is composed of four contigous fields. These fields are: the key length, the data length, the key, and the data. The key length is a one byte field which contains the length, in bytes, of the key field. The data length is another one byte field. It contains the length, in bytes, of the data field. The key is the field which is used to uniquely identify the record. The data field contains the user's data which is associated with the key.

The size of both length fields is one byte. The maximum integer which may be stored in a one byte field is 255; therefore the maximum length for the key field is 255 bytes and the maximum length for the data field is 255 bytes. Thus the maximum length for a data record is 512 bytes.

3.2.2 The Index Record

A KSAM index record is composed of four contigous fields. These fields are: the length, the displacement, the pointer to the son block, and the non-redundant part of the key.

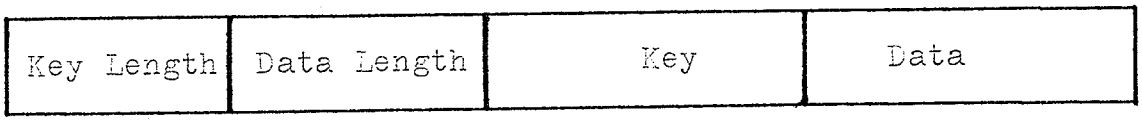


Figure 2: The KSAM Data Record

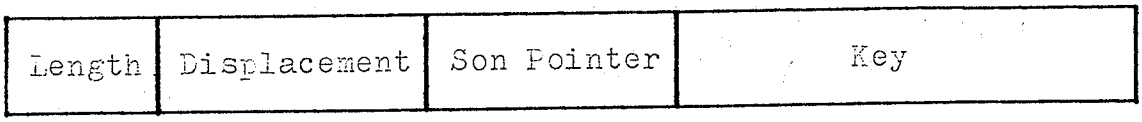


Figure 3: The KSAM Index Record

The length, displacement, and key fields are all related to one another. In order to conserve space within the index blocks a technique referred to as front key compression is employed. Using this technique the only part of the key that needs to be stored is that part of the key which is different from the previous key. For example, suppose that two consecutive keys are ABCD and ABCEF. The difference between these two keys is that the character D in the first key has been replaced by the characters EF in the second key; the characters ABC are the same in both records. In this example the only characters that need to be stored in the key field of the second record are the characters EF.

The length and the displacement fields contain the information required about where these letters should be placed to form the second key. The length field contains the number of characters to be placed; in this case the length field would have the value 2. The displacement field determines the number of characters from the start of the previous key to be retained. In this example the value of the displacement field would be 3 because three characters, ABC, from the previous key will be used.

The son pointer is an integer that forms a link from one level of the tree to the next level. The son block may be either an index block or a data block.

Alignment constraints complicate the manipulation of the son pointer. Because records are of variable length there is no method of ensuring that the son pointer will be at an even numbered address as the computer hardware requires without inserting unused bytes between records to cause proper alignment. In order to deal with this problem the son pointer is stored as two single bytes rather than as a single integer. When the son pointer is required the integer form of the son pointer must be re-created.

3.2.3 The High Value Index Record

The high value index record is defined to be the index record with the highest possible key. This key never appears in a data block, in fact, this key can not be formed by the user.

The main purpose of the high value index record is to ensure that all index block searches lead to another data or index block. Each index block search is terminated when an index record with a higher key than the search key is found. For every possible search key the high value key is greater. Thus, no index search ever proceeds past the high value record and therefore the index search always leads to some data block. The combination of these two results facilitates the search algorithm and the insert operation.

The other purpose of the high value index record is to enable the user to load the KSAM file in ascending order. By loading the file in ascending order the high value key will cause each record to be added to the file at the end of the last data block. This minimizes the amount of movement of the data within the data blocks. By adding records at the end of the last data block the KSAM file system ensures that the requested amount of free space is distributed within the file.

The high value index record has the format of any ordinary index record. To differentiate this record from other index records the length field is set to be 255 and the displacement is also set to be 255. By combining these two fields with these values it would seem to be possible to create a key with a length of 510 bytes. Because the maximum key length is only 255 bytes these values for the length and displacement fields clearly indicate that this record is the special record containing the high value key.

3.3 THE KSAM INFORMATION BLOCKS

3.3.1 The Free Space Count

The free space count is an integer found at the beginning of each type of information block. This integer, as the name implies, is used to keep a count of the number of unused bytes within the block.

The free space counter has a secondary purpose. This counter is also used to distinguish between index blocks and data blocks. In the root and in the index blocks the free space count has a value that is less than zero. In the data blocks the free space counter is greater than zero. In order to ensure that the free space counter never becomes equal to zero special processing must be undertaken. The space occupied by the null record which is required at the end of the pertinent information in both index and data blocks is included in the free space count even though that space is not free. This method of encoding the flag used to differentiate between types of blocks is efficient both from a storage point of view and from a processing point of view.

3.3.2 The Data Block

A KSAM data block contains: a positive free space counter, a series of data records, and a null record.

The data records are sorted, by key, into ascending alphanumeric order. The keys are of variable length; therefore two keys may be compared that are of unequal length. When keys of unequal length are compared the shorter of the two keys is considered to be extended with the lowest possible character to the length of the longer key. Using this method of comparison key AB precedes key AB0.

Figure 4: The KSAM Data Block

The null record consists of a single byte with the value zero. This byte is used to mark the end of the pertinent information in the block.

3.3.3 The Index Block

A KSAM index block contains: a negative free space counter, a series of index records, and a null record.

The first index record in an index block contains a complete key. In other words, the value of the displacement field of the first index record in the block will always be zero. Each subsequent record contains compressed keys sorted, by the original key, into ascending order.

The son pointer in each index record points to a data or an index block on the next level of the tree. The son block contains all records whose keys are greater in value than the key of the previous index record and whose keys are less than or equal in value to the key of the current index record.

The null record used to mark the termination of the pertinent information in the index block may be either the special high value index record or it may be an index record with a key length of zero. The high value index record functions as any other index record. The second type of null index record functions like the null record in a data block.