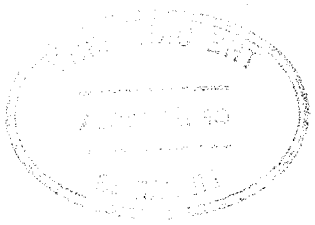
A Primary Key Retrieval System for the PDP-11

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

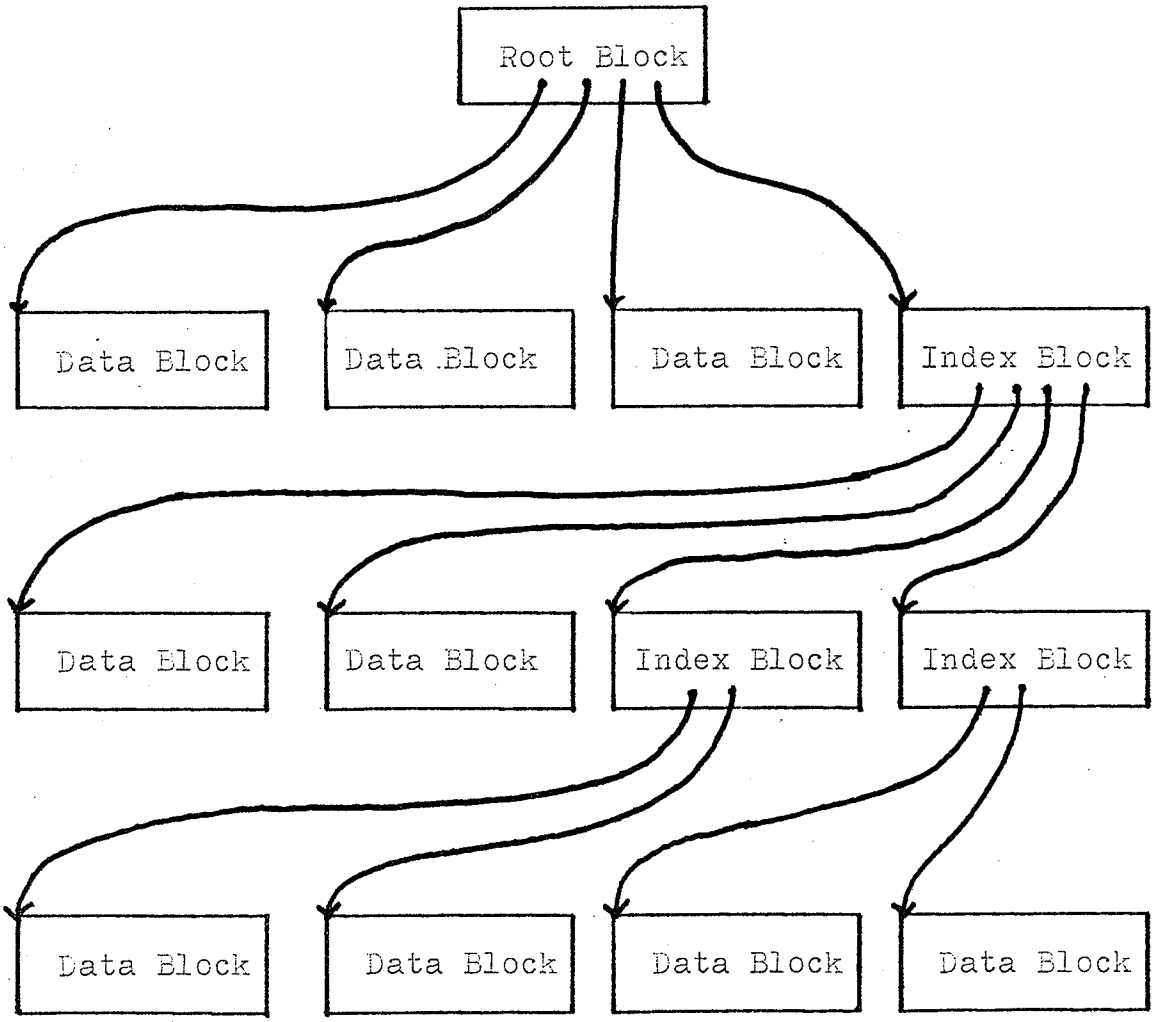


Figure 1: The KSAM Data Structure

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.

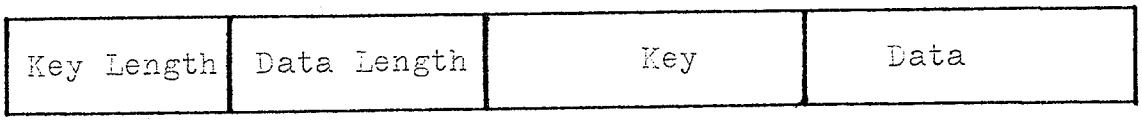


Figure 2: The KSAM Data Record

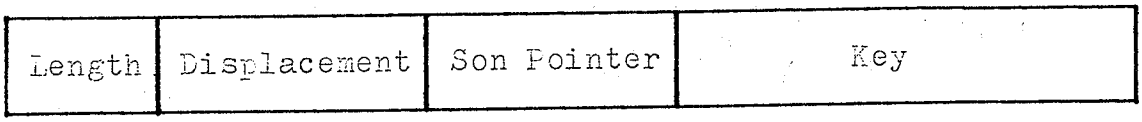


Figure 3: The KSAM Index Record

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.

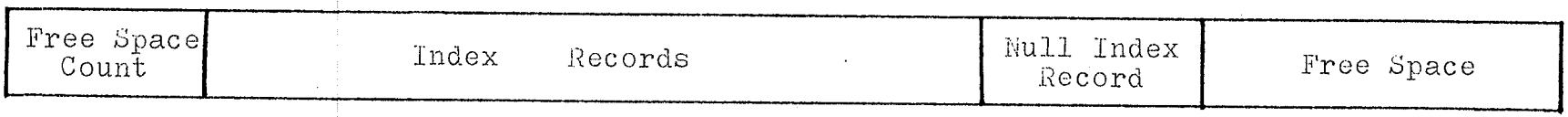


Figure 5: The KSAM Index Block

3.3.4 The Root Block

The KSAM root block is a special type of index block. It is distinguished from other index blocks by its position. The root block is always the first block in the KSAM file and it always has the block number zero. In addition to containing all of the information found in all index blocks and serving as an index block itself, the root block has four extra fields which are used throughout the KSAM system. These four extra fields are: the blocksize, the free chain pointer, the end of file block number, and a percentage indicating the amount of free space to be left as the file is loaded.

The blocksize is an integer which contains the size, in bytes, of all of the information blocks in the KSAM file. The blocksize must be a multiple of 512 greater than or equal in value to 1024. These contraints on the blocksize were chosen to facilitate the interaction between the KSAM file system and the UNIX operating system.

The free chain pointer is an integer which points to a linearly linked list of empty blocks. These blocks are blocks that have been emptied by the delete operation and returned to the system for reuse. In any situation where the system requires an empty block, this list is checked and the first block is reused before any other action is taken.

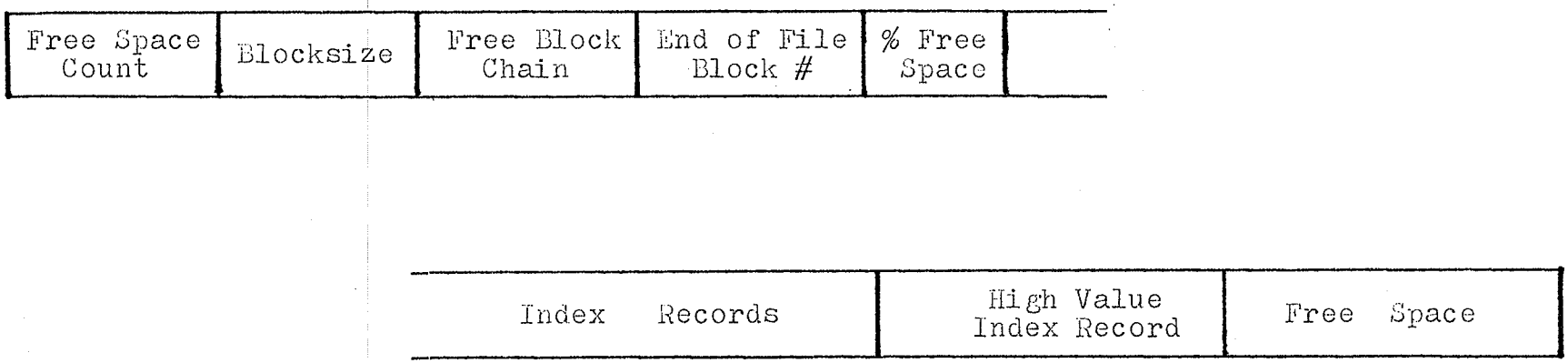


Figure 6: The KSAM Root Block

The end of file block number is the number that the next block after the last block in the KSAM file should have. The end of file block number is used to allocate a new block when a new block is required and an empty block is not available on the free block chain. In this case the file will be expanded by requesting that the UNIX operating system extend the KSAM file by adding a block after the last block in the KSAM file.

The final extra field in the root block is used to ensure that a certain amount of free space will be distributed throughout the KSAM file. This will allow for later insertions and changes in the KSAM file and is intended to reduce the number of data and index block splits required. The field contains an integer whose value is between 0 and 50. This value stands for the percentage of space to be left free in each data block when adding data records at the end of the KSAM file.

The root block will always have as its null record the special high value record. In all other characteristics the root block is organized and used in the same fashion as any other index block.

3.4 THE KSAM FILE PROCESSING STRUCTURE

3.4.1 The Internal Positional Pointers

Each KSAM file has associated with it a number of positional pointers. These pointers are created during the operation of opening the KSAM file for processing. A positional pointer is used to locate a data record within the KSAM file. The pointer is used primarily when processing the file sequentially.

The positional pointer is composed of three integral fields. These fields are: the block number, the offset from the beginning of the block, and a pointer to the index block which points to this particular data block.

The block number in the positional pointer refers to the current data block. The offset is the position of the current record given as the number of bytes from the start of the data block to the start of the record. The final field is used to indicate which index block is the immediate predecessor in the tree of the current data block.

3.4.2 The Buffering System

In order to reduce the number of input and output operations to the minimum a buffering system is used. Several information blocks are kept in main memory simultaneously. When a block is needed by the user the KSAM system first searches for the requested block among those blocks which it

already has in main memory. If the requested block is found then no input or output operation is required. The system works with the copy of the block in main memory. If the block that was requested is not found among the blocks in main memory then it will be read into main memory replacing a block that is currently there. The block to be replaced will be the block that was accessed least recently. If the block to be replaced has been modified since the last time it was written out then it will be copied to external storage before the new block is read in to replace it.

Chapter IV

INFORMATION RETRIEVAL

4.1 LOCATING A RECORD

The search for any data record begins with the first index record in the root block. The search key is compared to the key of the first index record in the root block. If the search key is greater than the key of the first record then the complete key of the second index record is created using the information in the second index record. The search key is then compared to the key of the second index record. This process is repeated until an index record is found that has a key greater than or equal in value to the search key. This search will always be successful because of the high value index record which is considered to have a key which is greater in value than any other key. After an index record is found that has a key greater than or equal in value to the search key the son pointer is used to find the block at the next level in the tree.

The next block down the tree may be either an index block or a data block. In the case of an index block, indicated by a free space count less than zero, the processing is the same as the processing in the root block. This sequence will be repeated through as many levels of index blocks as

required. Eventually the son block will be a data block, indicated by a positive free space count, and the search will continue at the data block level.

Within the data block a simple linear search for the requested record is performed either until the record is found or until the system recognizes that the record is nonexistant. The record is nonexistant if either the null data record at the end of the block is encountered or if a record with a key greater in value than the search key is found.

The location pointer is set regardless of whether the search record is found or not. If the record is found then the pointer identifies the start of the record. When the record is not found the location pointer identifies the record with the next higher key.

4.2 LOCATING THE NEXT SEQUENTIAL RECORD

The POINTNEXT routine is used to advance an internal pointer from one data record to the data record which follows it sequentially. This routine will be used whenever the KSAM file is processed sequentially.

If the next sequential record is within the same data block then the pointer may be advanced by changing the pointer's offset. The offset will be incremented by the total length of the current record. The record which follows the

current record physically will be the next record sequentially.

If the current record is the final record in the data block then the father index block will be used. The father index block is searched for the index record that points to the current data block. As this search is being carried out the complete key for the index record is being re-created. If the index record pointing to the current data block is not the high value index record then the next highest possible key is created and then searched for in the usual manner. Whatever record is found as a result of this search will be the record with the next highest key. In the case where the index record of the current data block is the high value index record then the current data record is the last record in the file and the POINTNEXT routine will indicate that the end of the file has been reached.

Chapter V

MODIFYING THE KSAM FILE

5.1 INSERTING A RECORD

Data records are added to the KSAM file by the use of the insert operation. The insert operation comprises both insertion in the middle of a block and also appending to the end, used initially to load the KSAM file. The insert routine is capable of recognizing the type of operation to be performed by noting where within the KSAM file the new record is to be added.

All insert operations begin with a search for the record that is to be inserted. If this record is found to already exist within the file then an error has occurred and the insert process will be terminated. If the record is not found within the file then the insert may take place. It should be noted that the search operation leaves the pointer set to the location where that record should be inserted.

If the search for the insert record was terminated by finding a record with a higher key than the record to be inserted then the insert takes place within the data records in the data block. The total length of the record to be inserted is compared to the amount of free space available

within the data block. If insufficient free space remains then a block split is required before the insert operation may be completed. If sufficient free space is available within the data block then the insert operation may procede. All records in the data block that are above the insert point, including the null data record at the end of the block, are shifted to the right by the length of the data record to be inserted. The new data record is then copied into the data block starting at the insert point. The free space counter is updated to reflect the change in the amount of free space available.

If the search for the insert record was terminated by finding the null data record at the end of the data block then the insert takes place at the end of the block. This case is referred to as the load or append case. This operation differs from the ordinary form of the insert operation only with regards to the amount of free space available. The amount of free space available is reduced by the number of bytes given by the percentage specified when the file was created. This ensures that approximately the specified percentage of free space is left in every data block within the file.

The insert operation sets the internal position pointer. The positional pointer specified in the call to the insert

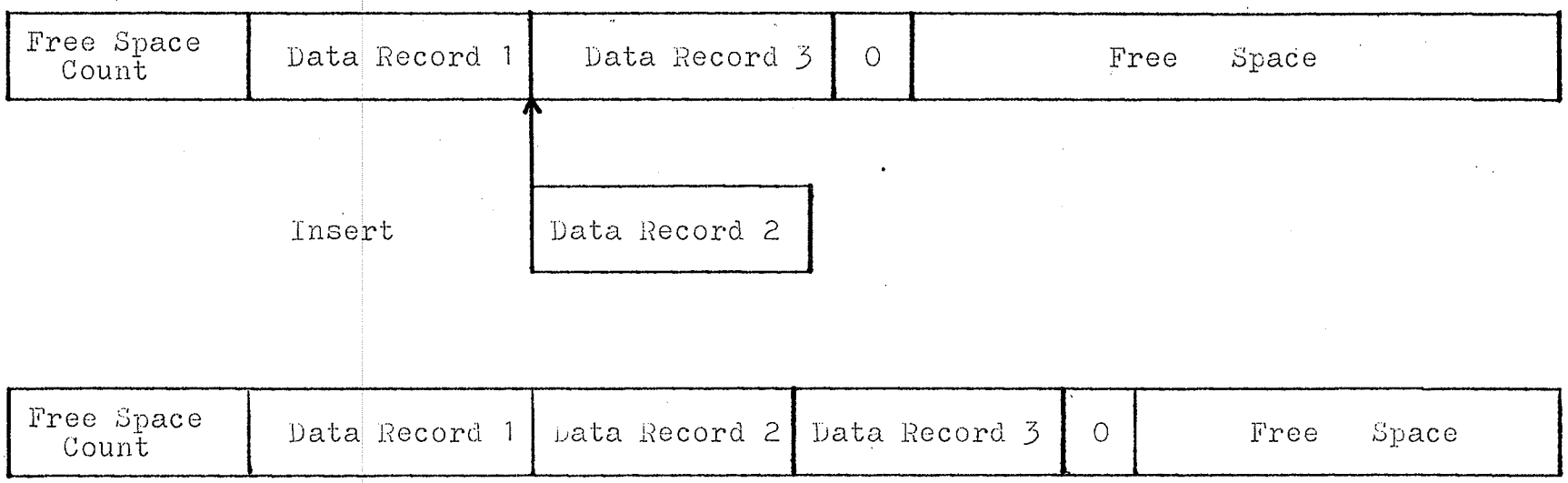


Figure 7: Inserting a Data Record

routine is always set to the record that was inserted. All other internal pointers are left pointing at the records that they pointed to before the insert operation was carried out.

If necessary internal pointers are changed to reflect the new data structure to ensure that they point to the same records after the insert as they did prior to the insert operation.

One final difference between the two types of insert operations should be noted. When the blocks are split because there is insufficient free space available the split point is selected in a different manner for each type of insert. When appending a record the split point is selected so that the current data block will remain unchanged and the new block, where the insert will take place, will be empty. When a record is inserted into the middle of a data block and a split is required the split point is selected so that the current data block will be split into two parts which will be approximately equal in size after the record is inserted.

5.2 DELETING A RECORD

The delete operation is used to remove data records from the KSAM file. In order to delete a data record a search for a record with the proper key must be undertaken. If that

record is not found then the record does not exist within the KSAM file and therefore an error has occurred. If the record exists within the KSAM file then the delete operation may be carried out.

The amount of free space in the current data block is increased by the total length of the delete record. All records above the delete point, including the null data record at the end of the data block, are moved to the left by the length of the deleted record. This shift overlays the deleted record and thus eliminates it from the data block.

It is possible that after a series of delete operations that a data block may be left that contains no data records other than the null data record at the end of the data block. In this situation the data block no longer performs a useful function. This empty data block will be added to the free block chain which is pointed to, in the root block, by the free chain pointer. In order to carry out this operation the index record pointing to this data block must be deleted from the father index block. The father index block is searched until the index record pointing to the current data block is found. If an index record follows the index record to be deleted it is modified so that it will follow the index record prior to the index record being deleted. The index record pointing to the current data block is deleted by shifting all records above it, including the null index

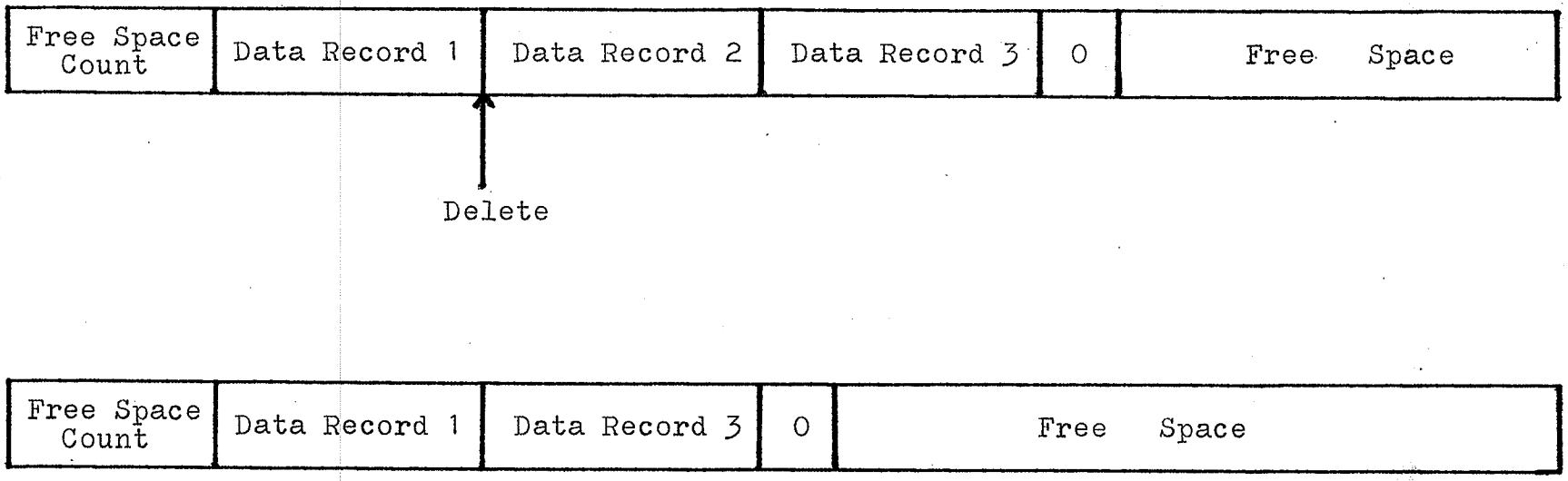
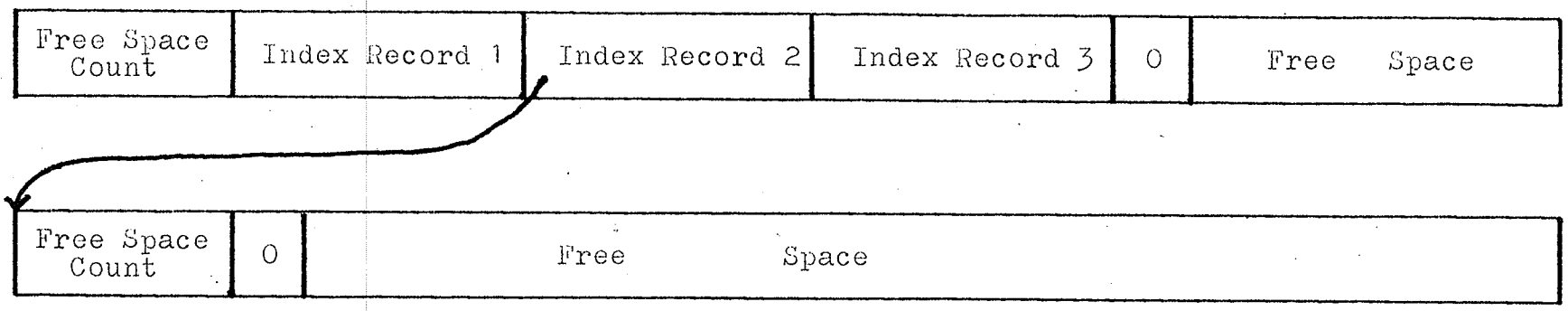


Figure 8: Deleting a Data Record



The data block is added to the free block chain to produce:

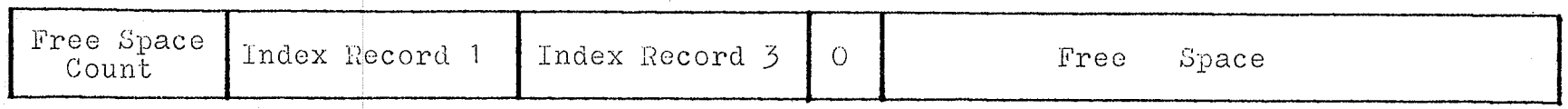
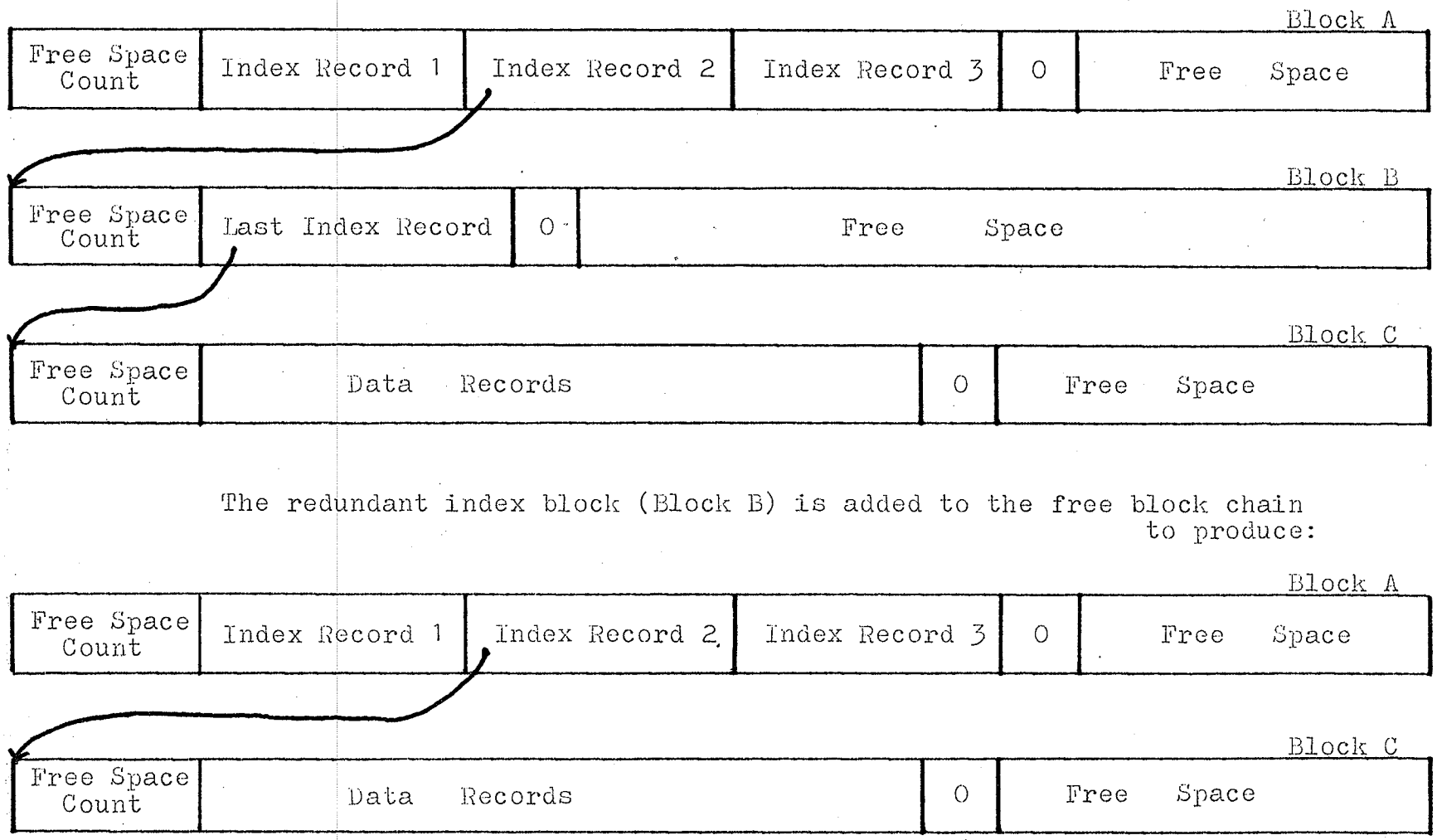


Figure 9: Freeing an Empty Data Block



The redundant index block (Block B) is added to the free block chain to produce:

Figure 10: Freeing a Redundant Index Block

record at the end of the index block, to the left. Once the index record that is pointing to the now empty data block is deleted then the current data block may be added to the free block chain.

If any delete operation leaves an index block with only a single index record in it, then that index block is no longer necessary and it may be added to the free block chain. In order to carry this out the index record for this index block must be located and updated. The father block for this index block is searched until the index record pointing to the unnecessary index block is found. The son pointer of the index record to be updated is changed to the son pointer of the last remaining index record in the unnecessary index block. Once this change has been made then the unnecessary index block may be added to the free block chain.

At the conclusion of the delete operation the pointer specified in the call to the delete routine is left pointing to the data record which follows the deleted record sequentially. If the last record of a block is deleted then the pointer is changed to identify the next sequential record. All other internal pointers are updated to ensure that they point to the same records after the delete operation as they pointed to before the delete operation.

5.3 REWRITING A RECORD

The rewrite routine is used to make changes to the data portion of a data record. This routine may change the contents of the data portion of the data record and it may change the length of the data portion of a data record. This routine may not be used to modify the key portion of a data record in any way.

In order to rewrite a data record the record must already exist within the KSAM file. The rewrite record is searched for in the usual manner and if it is not found then an error has occurred. If the rewrite record is found then the rewrite operation may take place.

The length of the new data portion is compared with the length of the current data portion of the data record. If the length of the new data portion is less than or equal to the length of the current data portion then the rewrite is a relatively straightforward operation. If the length of the new data portion is longer than the length of the current data portion then the rewrite operation is a more complex task.

In the case where the length of the new data portion is less than or equal to the length of the current data portion then the existing data portion is overwritten by the new data portion. If the length of the new data portion is less than the old data portion then the difference is added to

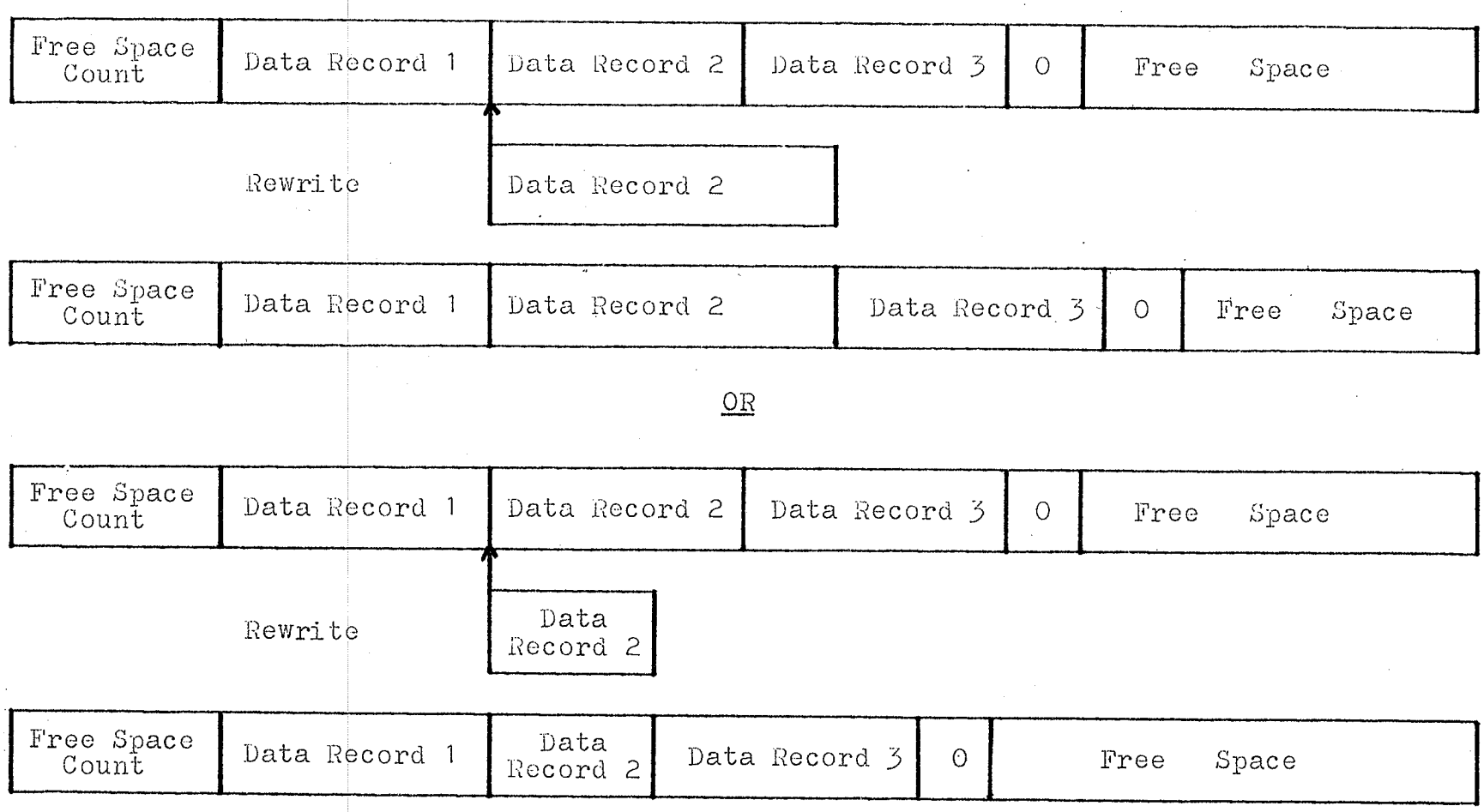


Figure 11: Rewriting a Data Record

the free space count and all records above the rewrite point are shifted to the left by the number of bytes freed.

In the case where the length of the new data portion is greater than the length of the existing data portion then the difference between these lengths is compared to the amount of free space available within the data block. If there is sufficient free space then all data records above the rewrite point are shifted to the right by the difference in the number of bytes between the new data portion and the old data portion. After the records have been shifted then the new data portion is written into the space created by the shift. The free space counter is then decremented by the difference between the lengths of the new and the old data portions. If a data block split is required then the split is made and then this process is undertaken.

At the conclusion of a rewrite operation the pointer specified in the call to the rewrite routine is left pointing to the data record that was updated. All other internal pointers are updated to ensure that they point to the same data records after the rewrite operation as they did before the rewrite operation.

Chapter VI

INFORMATION BLOCK SPLITS

6.1 DATA BLOCK SPLITS

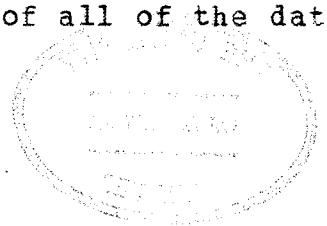
Data block splits are required whenever either an insert or rewrite operation can not be completed due to insufficient free space remaining in the data block. During the course of a data block split operation a data block is added to the KSAM file. Some of the data records that are in the data block to be split are moved into the new data block. This operation will leave more free space in the data block that was split. The free space in the data block has increased therefore the insert or rewrite operation that is required can now be completed. This split point is selected so that the exact midpoint of the data to be split will be located within the first record to be moved into the new data block.

In order to split a data block into two parts the first consideration must be the selection of the point at which the data block is to be split. This split point will be at the start of the first data record to be moved into the new data block. The selection of the split point occurs in two different ways.

The first method of selecting the point at which the data block is to be split is used in the append form of the insert operation. In this situation the data record is to be inserted at the end of the data block and the split point is selected so that the data record to be inserted will be the first record in the new data block and the existing data block will be left unchanged. The data block to be split does not change but the index record pointing to the split block will be modified to reflect the new situation.

The second method of selecting the point at which the data block is to be split is used in the insert and the rewrite operations. The split point is selected so that the two data blocks will have approximately the same amount of free space in each after the insert or rewrite operation is completed.

The operation of splitting a data block into two parts requires that the father index block be modified. A new index record must be created to point to the split block. This index record will have as its key the key of the data record immediately prior to the split point. A second new index record must be created to point to the new data block. The key for this index record will be the key that was in the index record that pointed to the split block before the data block split was required. This will ensure that the keys of all of the data records in the split block are less



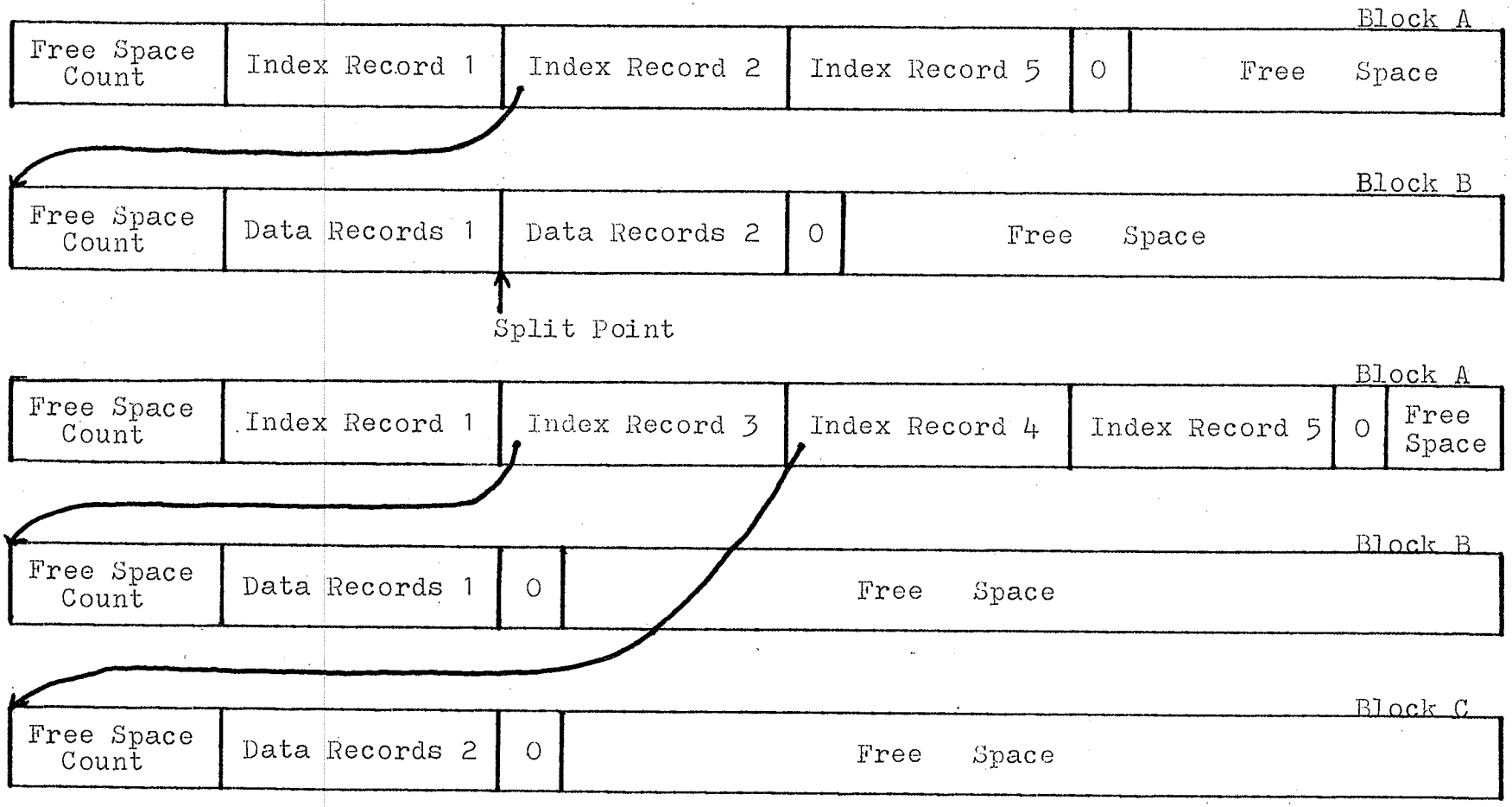


Figure 12: The Data Block Split

than or equal to the key of the index record pointing to the split block. It will also ensure that the keys of the data records in the new data block will be less than or equal to the key of the index record pointing to the new data block. These two index records are created and if there is sufficient free space available to accomodate the new records in the father index block then the data block split operation may be completed. If there is not sufficient free space available in the father index block for the two new index records then the father index block must be split before the data block split can be completed.

The actual operation of splitting a data block is divided into three parts. Firstly, the new data block must be created. Secondly, the father index block must be modified by the creation and insertion of the two new index records. Finally, the split block must be updated to indicate the new end of the data records and the new amount of free space available.

In order to create a new data block a block must either be obtained from the free block chain or added to the end of the KSAM file. The new data block is loaded with copies of all of the data records in the split block that are found above the split point. In the case of the append form of the insert operation there are no data records above the split point except the null data record at the end of the split

data block. The free space counter of the new block is calculated and written into the new data block. At the conclusion of the creation of the new data block the new data block is written onto the external storage device where the KSAM file is stored.

The father index block is updated by replacing the index record that points to the split block with the two new index records created for the split data block and the new data block. All other index records in the father index block will be shifted if necessary to accomodate the new index records. The free space counter in the father index block will be updated and the father index block will be written onto the external storage device.

To complete the data block split the split block itself must be updated. The new value for the free space counter is calculated and written into the split block. A null record is inserted into the split data block at the new end of the data block, the split point. When the split data block has been updated it will be written onto the external storage device.

The final operation involved in the data block splitting process concerns the internal positional pointers. All of the internal positional pointers are tested and updated if necessary to ensure that they point to the same data records after the data block split as they pointed to before the data block split.

The data block splitting process is now complete and the original insert or rewrite request may be completed. Because data records are at most 512 bytes in length and blocks are at least 1024 bytes long a split ensures that enough space will be made available for the record.

6.2 INDEX BLOCK SPLITS

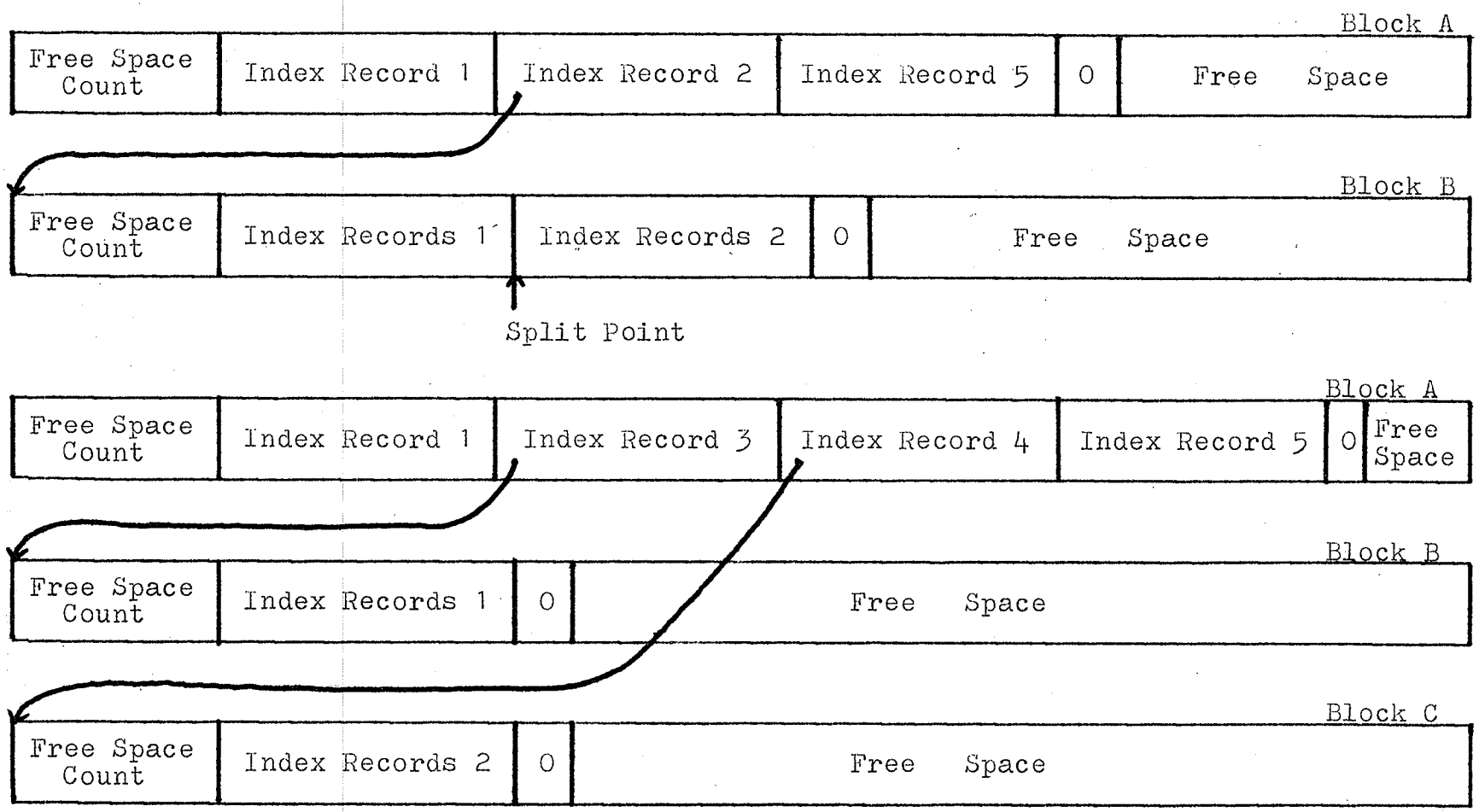
An index block split is required when an attempt to split a son data or index block fails because there is insufficient free space remaining in the current index block to accomodate the two new index records required. During the course of an index block split a new index block is added to the KSAM file. Some of the index records that are in the index block to be split are moved into the new index block. This operation increases the amount of free space remaining in the split index block. The increase in the amount of free space available in the split index block will allow sufficient free space to accomodate the two new index records required by the splitting process for the splitting of the son block.

The selection of the split point in the case of an index block split is a much simpler situation than in the case of a data block split. Only one method of selecting the point at which the index block is to be split is used. The split point is selected so that the split index block and the new index block will have approximately the same amount of free

space available in them after the two new index records are inserted.

The index block splitting operation requires that the father index block be modified by replacing the index record that points to the split block by two new index records. The second of the two new index records will have the same key as the index record which currently points to the split index block. The son pointer of the second of the two new index records will point to the new index block. A totally new index record must be created for the first of the two new index records. The first new index record will have as its key the key of the index record immediately prior to the split point in the split index block. This first new index record will point to the split index block.

This method of creating the two new index records will ensure two things. The keys of all of the index records in the split index block are less than or equal to the key of the index record pointing to the split index block. This method also ensures that the keys of all of the index records in the new index block are less than or equal to the key of the index record pointing to the new index block. These two new index records are created and if sufficient free space remains in the father index block to accomodate the new records then the index block splitting process can procede. If there is not sufficient free space available in



Note: The first index record in block C will contain a complete key.

Figure 13: The Index Block Split

the father index block to accomodate the two new index records then the father index block must be split before the index block split can be completed.

The actual index block splitting process can be subdivided into three parts. The first part of the index block splitting process is the creation and loading of the new index block. The second part of the index block splitting process is the modification of the father index block by the insertion of the two new index records. The final phase of the index block splitting process is the modification of the split index block to indicate the end of the index records in the index block and the amount of free space available.

The creation of a new index block must be begun with the allocation of an empty information block. This empty information block is either obtained from the free block chain or is appended to the KSAM file. The first index record in any index block must contain the complete key for that record. The complete key for the first index record in the new index block is created using the complete key of the last record to remain in the split index block and the information to be found in the index record that immediately follows the split point in the split index block. After the first index record is written into the new index block then the remaining index records above the split point are copied

into the new index block. The free space counter for the new index block is calculated and written into the new index block. Once the new index block contains all of the necessary information it is written onto the external storage device which contains the KSAM file.

The father index block is updated by the replacement of the index record which points to the split index block by the two new index records pointing to the split index block and the new index block. The free space counter in the father index block is updated and the father index block is written onto the external storage device.

The final stage in the index block splitting process is the modification of the split index block. The free space counter is calculated and written into the split index block. The null index record is inserted into the split block at the new end of the index block, the split point. At the conclusion of the modification of the split index block it is written onto the external storage device.

The index block splitting process is now complete. The data or index block split required at the next level down the tree may now be completed.

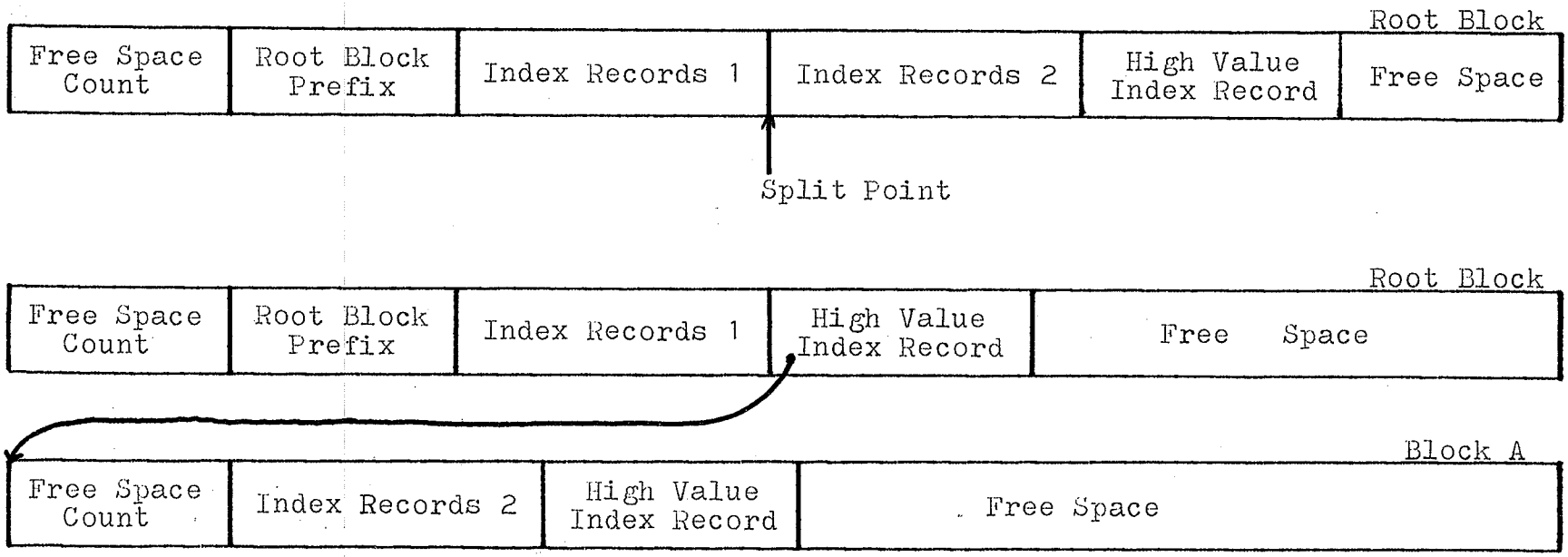
6.3 ROOT BLOCK SPLITS

A root block split is required when an attempt to split a son data or index block has failed because there is insufficient free space remaining in the root block to accomodate the two new index records required. During the root splitting operation a new index block is added to the KSAM file. Approximately half of the index records in the root block are moved into the new index block. This operation will increase the amount of free space available in the root block. The root block split also adds another level of index blocks to the KSAM file. At the conclusion of the root block split the split required at the next level of the tree can be performed.

As with all block splits the first step in the splitting of the root block is the selection of the point at which the root block is to be split. The split point is selected in the same manner as the split point is selected for any other index block.

The root block is the topmost block in the KSAM file. It does not have an index record that points to it therefore there is no father index block to be modified and no new index records to be created.

The operation of splitting the root block has only two parts to it. The first phase of the root block split is the creation and loading of a new index block. The second and



Note: The first index record in block A will contain a complete key.

Figure 14: The Root Block Split

last phase of the root block split is the modification of the root block.

The actual splitting process for the root block follows the same algorithm as that followed for the splitting of an ordinary index block. A new index block will be created, filled in, and written onto external storage.

The last stage in the root block splitting process is the modification of the root block. The free space counter is calculated and written into the root block. The high value index record is used as the null index record in the root block. This record is inserted into the root block at the split point. The son pointer of the high value index record will be set to point at the new index block.

The high value index record may appear several times within a KSAM file. On each level of the index the high value record will be the rightmost index record. This arrangement of the high value index record occurrences will form a path from the root block down the right side of the tree to the block which contains the data record with the highest key in the KSAM file. This arrangement ensures that all index block searches will be successful and that all search operations will terminate in a data block. This arrangement of the high value record also facilitates the append form of the insert operation.

The root block splitting process is now complete. The data or index block split required at the next level of the tree may now be completed.

6.4 MULTILEVEL SPLITS

A multilevel split occurrs whenever more than one information block must be split in order to complete an insert or rewrite request. In this situation the block that is nearest the root block and requires splitting will be split first, then its son block, and so forth until eventually the data block is split and the insert or rewrite operation can be completed. For example, in the case where the root block, an index block, and a data block all require splitting the order in which the blocks would be split is: root, then index, and finally the data block.

Chapter VII

PROGRAMMING THE KSAM RECORD MANAGEMENT SYSTEM

7.1 GENERAL DESCRIPTION OF THE KSAM ROUTINES

KSAM is written in C Language and is designed to be used on the PDP-11 with the UNIX operating system. It occupies approximately 15K bytes of memory. The KSAM record management system takes advantage of many of the special features available when using the UNIX operating system. For example, many of the possible errors that can be made by the KSAM user are detected by the UNIX operating system which will either correct the error or provide diagnostic messages and codes for the programmer's use. The UNIX operating system also places a few restrictions on the user. The blocksize best suited to UNIX is 512 bytes therefore the KSAM system enforces that optimal blocksize, or a multiple of it, on the system user. The flexibility and ease of use of the UNIX operating system makes it ideally suited to the KSAM record management system.

The KSAM record management system is composed of thirteen routines. These thirteen routines are divided into two categories. In one category are those routines which are used to manipulate the KSAM file as a whole. The routines which operate on the entire KSAM file as a single entity are:

KCREATE, KOPEN, KCLOSE, and KPURGE. The second category of KSAM routines are those routines which manipulate either single data records or single positional pointers. The routines included in the second category are: KPOINT, KRESET, KPOINTN, KRWRITE, KINSERT, KDELETE, KREAD, KREADN, and KREADK. This second category is further subdivided into three parts: the routines which locate a record, the routines which modify the file, and the routines which are used to retrieve a record.

7.2 KSAM RETURN CODES

There are a number of codes which are returned as the value for the various KSAM routines used within the KSAM record management system. There are five codes that may be encountered by the user and these codes are called user codes.

Every routine in the KSAM record management system is a user callable function. This implies that each of the KSAM routines returns a value. With the exception of the KOPEN routine which returns a pointer to a control block which it creates when it terminates successfully all KSAM routines return one of the five possible integral codes.

7.2.1 User Code -1

The user code -1 is used to indicate that some sort of unrecoverable error has been detected by the UNIX operating system. By itself this is not sufficient information to form any sort of a meaningful diagnosis of the error. UNIX maintains a system of error handling that is very useful in this situation. The UNIX error system sets the global variable ERRNO to an error code of its own. This code may be used to determine the nature of the error and corrective action may be taken. In practice, this code should only be encountered when using the routines: KCREATE, KOPEN, and KCLOSE. Any other occurrence of the code -1 indicates an error in the KSAM system itself. In this situation the error causing program should be examined by a systems programmer for the necessary corrective action.

7.2.2 User Code 0

The user code of 0 is used by all routines except KOPEN to indicate that the requested operation has been carried out successfully.

7.2.3 User Code 1

The user code of 1 is used to indicate that the requested data record does not exist within the KSAM file.

7.2.4 User Code 2

The user code of 2 is used to indicate that the requested data record does not exist within the KSAM file. This code further indicates that the requested data record is beyond the end of the file.

7.2.5 User Code 3

The user code of 3 is only returned by the KINSERT routine. The code 3 is used to indicate that a data record can not be inserted into the KSAM file because it already exists within the file.

7.3 KSAM FILE MANIPULATION ROUTINES

7.3.1 KCREATE

The KCREATE routine is used to create a KSAM file. This routine is also used to specify the attributes of the file. The KCREATE routine accepts three arguments: a pointer to the UNIX name of the file to be created, an integer containing the size of the information blocks to be used, and a second integer specifying the amount of free space to be left in each data block during the initial loading of the file.

The name of the file must be in standard UNIX format. The fully qualified name is stored as a character string and a pointer to this string is used as the parameter to the KCREATE routine.

The second parameter is an integer that specifies the blocksize to be used in the creation of the KSAM file. This blocksize must be a multiple of 512 greater than or equal to 1024. If an incorrect value is specified 1024 is used.

The third and final parameter is another integer used to specify what percentage of each data block should be left free when a data record or data records are appended to the file. This integer must be between 0 and 50. Any incorrect value for this argument is assumed to be 0. It is recommended that a non-zero number be specified for this parameter. If no free space is left in the data blocks then later modifications of the file will result in a large number of data block splits.

The KCREATE routine creates a UNIX file with the name specified by the first argument. This file appears to the UNIX operating system to be the same as any other file. This will allow the programmer to use the UNIX system routines for dumping, moving, and copying the KSAM file. The user is cautioned that the KSAM file should not be modified with any routines other than the KSAM record management system.

The KCREATE routine creates and writes into the KSAM file the first two blocks of the file. The first of these two blocks will be the root block. The second block will be the first data block of the file. The fields in the prefix of the root block are filled in in the following manner: The

blocksize is taken from the parameter list and is written into the root block prefix. In the same way the percentage of free space to be left is copied into the root block prefix. The free block chain is marked to indicate that the free block chain is empty. The end of file block is set to 2 indicating that the next block past the end of the file is block number 2. The final piece of information inserted into the root block is the high value record. This index record is set to point to the last data block in the KSAM file, block number 1. The free space count is calculated and the root block is written onto the external storage device where the KSAM file will reside. The first data block, block number 1, is marked with a null record to indicate that the block is entirely empty. The free space count is filled in and the block is written onto the external storage device.

7.3.2 KOPEN

The KOPEN routine is used to prepare a KSAM file for processing. The routine performs two functions: the opening of the KSAM file for both input and output and the creation of the control block that will be needed for all subsequent calls to the KSAM routines.

The KOPEN routine requires three arguments: a pointer to the character string which contains the same UNIX file name as specified when the file was created, an integer specifying the number of internal position pointers to be created,

and a second integer specifying the number of input/output buffers to be created.

The name of the file must be in standard UNIX format. The fully qualified name is stored as a character string and a pointer to this string is used as the parameter to the KOPEN routine. This name must be be the same name that was used by the KCREATE routine to create the file.

The number of internal position pointers to be created must be a positive integer greater than or equal to 1. If an incorrect argument is detected then the value is assumed to be 1.

The third parameter to the KOPEN routine is a second integer which specifies the number of input/output buffers to be allocated. This parameter must be a positive integer greater than or equal to 2. If an incorrect argument is supplied then the KOPEN routine assumes that the number of buffers required is 2.

The KOPEN routine opens the UNIX file specified for both input and output. This mode of communication allows the file to be both read and updated at the same time.

The KOPEN routine creates the number of internal position pointers requested. These pointers are included in the control block created by the KOPEN routine.

The KOPEN routine is also responsible for the creation of the input/output buffers required. In the same way as the internal position pointers the input/output buffers are included in the control block created by KOPEN.

The control block created by the KOPEN routine contains fields that are required by all of the other KSAM routines except, of course, KCREATE. The information in this control block is extracted from the root block and when combined with positional pointers and input/output buffers provides all of the information required to process the KSAM file.

The KOPEN routine returns one of two values. The first value is the address of the control block created. This value must be stored and provided when calling the other KSAM routines. When a valid address is returned this indicates that the KOPEN routine has opened the file and created the control block successfully. A return code of -1 is used to indicate that an unacceptable UNIX error has occurred. In this case the UNIX error handling routines should be used. The most commonly occurring errors are: trying to open a file that does not exist, trying to open as a KSAM file a file that is not a KSAM file, and attempting to open a file that is already open.

7.3.3 KCLOSE

The KCLOSE routine is used to end the processing of a KSAM file. This routine requires only one argument, the pointer to the control block created by the KOPEN routine.

The KCLOSE routine examines all of the input/output buffers in the control block. If any of these buffers have been changed since the last time that they were written onto the external storage device where the KSAM file resides then they are written out into the KSAM file. After all necessary output has been completed the UNIX file containing the KSAM file is closed. The final operation performed by the KCLOSE routine is the release of the space occupied by the control block created by the KOPEN routine.

7.3.4 KPURGE

The KPURGE routine is used to force the KSAM record management system to write out onto the external storage device any input/output buffers that have been changed since the last time they were written out into the KSAM file. This routine requires only one argument, the pointer to the control block created by the KOPEN routine.

The KPURGE routine may be called at anytime during the processing of the KSAM file. This routine will ensure that the version of the file on the external storage device is completely up to date with the version of the file in main

memory. In the case of a power or system failure this will reduce the amount of work that will have to be redone.

7.4 KSAM RECORD AND POINTER MANIPULATION ROUTINES

7.4.1 Record Location Routines

7.4.1.1 KPOINT

The KPOINT routine is used to locate a particular record or the location where that record should appear. This routine sets the specified pointer to the desired record. This routine requires three arguments: the pointer to the control block created by the KOPEN routine, an integer containing the number of the pointer to be set, and a pointer to the key of the record to be searched for.

The pointer number indicates which of the positional pointers is to be set to the location of the sought after record. If the record is found then the positional pointer is set to the start of that record. If the search record is not found then the positional pointer is set to point to the record with the next higher key.

The third parameter is a pointer to the key of the desired record. This key is stored in two contigous parts: the first part is a one byte field which contains the length of the key; the second part is a series of bytes containing the actual key.

The point routine conducts a search for the requested record. This search begins in the root block and procedes down through the tree until the data block which should contain the requested record is located. Once the data block has been located a linear search for the requested record is initiated. If the record is found then the pointer is set to point to the start of the record. If the record is not found then the pointer is set to the record with the next higher key.

7.4.1.2 KRESET

The KRESET routine is used to set a positional pointer to the first record in the KSAM file. This routine requires two arguments: the pointer to the control block created by the KOPEN routine and an integer containing the number of the pointer to be reset.

The KRESET routine resets the required pointer by conducting a search for the record with the least possible key. Whatever record is located by the search will be the first record in the KSAM file.

7.4.1.3 KPOINTN

The KPOINTN routine is used to advance a positional pointer from the record that it is currently pointing at to the next sequential record. The routine requires two arguments: the pointer to the control block created by the KOPEN rout-

ine and an integer containing the number of the pointer to be advanced.

7.4.2 Record Modifying Routines

7.4.2.1 KRWRITE

The KRWRITE routine is used to update a data record in a KSAM file. This routine requires three arguments: the pointer to the control block created by KOPEN, an integer containing the number of the pointer to be set, and a pointer to the new version of the record.

The new version of the record is composed of four contigous fields. These fields are: a one byte field for the key length, a one byte field for the length of the data field, the key, and the data.

The routine uses the key portion of the new version of the record to search for the record in the KSAM file. If the record is found then the data portion of the existing record is replaced by the data portion of the new version of the record. The other data records in the data block are shifted to make room for the new version of the data portion or to reuse space that is left free by the new version of the data portion. If there is insufficient free space available in the data block for the new version of the data portion then the data block is split and then the rewrite operation is completed. After the completion of the rewrite operation all

of the positional pointers in the file are examined and adjusted if necessary to ensure that all of the pointers point to the same records after the rewrite as they pointed to before the rewrite.

7.4.2.2 KINSERT

The KINSERT routine is used to add a data record to a KSAM file. This routine requires three arguments: the pointer to the control block created by the KOPEN routine, an integer containing the number of the pointer to be set, and a pointer to the data record to be inserted.

The data record to be added to the KSAM file is composed of four contigous fields. These fields are: a one byte field for the length of the key, a one byte field for the length of the data field, the key, and the data.

The KINSERT routine uses the key portion to search for the record to be inserted. If the record is not found then the insertion may proceed. The other data records in the data block into which the new record is to be inserted are shifted to allow room for the new record. The new record is then inserted into the data block. If there is insufficient free space available in the data block for the insert to take place then the data block is split and then the insert takes place. The pointer specified is set to point to the record just added to the file. All other positional poin-

ters are examined and if necessary adjusted to ensure that they point to the same data records after the insert as they pointed to before the new record was added.

In the case where the insertion is at the end of the file, the append case, then the free space to be left within tha data block must be at least as large as the percentage requested when the file was created. If this is not possible then a new block is added to the end of the file and the new record is inserted into the new data block.

7.4.2.3 KDELETE

The KDELETE routine is used to remove a data record from a KSAM file. This routine requires three arguments: the pointer to the control block created by the KOPEN routine, an integer containing the number of the pointer to be set, and a pointer to the key of the record to be deleted.

The key of the record to be deleted is stored as two contigous fields. These fields are: a one byte field containing the length of the key and a series of bytes containing the actual key.

The KDELETE routine searches for the delete record using the supplied key. If the record is found then it is removed from the KSAM file by shifting the other data records in the block to recover the space formerly occupied by the deleted record. If the data block is emptied by the delete operation

and/or an index block is made redundant then these blocks are freed and added to the free block chain. The pointer is set to point to the record which sequentially follows the data record just deleted. All of the other positional pointers are tested and adjusted if required so that they point to the same records after the delete as they did before the delete operation.

7.4.3 Record Retrieval Routines

7.4.3.1 KREAD

The KREAD routine is used to copy a data record in the KSAM file into a user supplied data area. This routine requires three arguments: the pointer to the control block created by the KOPEN routine, an integer containing the number of the pointer to the record to be copied, and the address of the data area into which the record is to be copied.

The KREAD routine uses the pointer to locate the required data record. The routine then copies the record into the area specified. The record is copied in the form of a one byte field for the key length, a one byte field for the length of the data portion, the key, and the data.

7.4.3.2 KREADN

The KREADN routine is used to advance the given positional pointer from the current data record to the next sequential data record and to copy the next sequential data record into the user supplied data area. This routine requires three arguments: the pointer to the control block created by the KOPEN routine, an integer containing the number of the pointer to be advanced to find the requested record, and the address of the data area into which the record is to be copied.

The KREADN routine uses the given pointer to locate the requested data record. The given pointer is advanced to the next sequential data record. If the next record exists then it is copied into the specified data area in the form of a one byte length field for the key, a one byte field for the data length, the key, and the data. The pointer is left pointing at the record copied.

7.4.3.3 KREADK

The KREADK routine is used to locate a record by key and to copy it into a user supplied data area. The routine requires three arguments: the pointer to the control block created by the KOPEN routine, an integer specifying the number of the pointer to be set, and the address where the key of the requested record is to be found and to where the complete record and key is to be copied.

The key of the requested record is in the form of a one byte key length followed by the actual key.

The KREADK routine uses the key supplied to search for the requested data record. If the record is found then the pointer is set to the start of the requested record. The data record is then copied into the supplied data area in the form of a one byte key length, a one byte data length, the key, and the data.

Chapter VIII

CONCLUSION

The KSAM record management system was designed to satisfy the basic requirements of a primary key retrieval system. The KSAM record management system is capable of identifying each data record by a single unique key field. This key is used to insert, delete, update, and search for any particular data record. A primary key retrieval system must have the capability of adding data records to the file and the KSAM routine, KINSERT, provides this facility. A primary key retrieval system requires the ability to delete data records from the file; The KSAM routine, KDELETE, may be used to remove data records from the KSAM file. The ability to update data records within the file is a requirement of a primary key retrieval system. This ability is provided by the KSAM routine, KRWRITE. The retrieval capabilities of the primary key retrieval system are provided by the six KSAM routines: KPOINT, KPOINTN, KRESET, KREAD, KREADN, and KREADK. The KSAM record management system provides all of the basic requirements of a primary key retrieval system.

A primary key retrieval system may, optionally, provide sequential data processing support. The KSAM record management system is designed to support sequential data process-

ing. Sequential data processing is enhanced by the organization of the data records within the KSAM data blocks. The data records are sorted, by key, into ascending order within each data block. This method of storing the data records ensures that, for most of the data records, the next record physically is also the next record sequentially. This provides for an easy method of progressing from one record to the next sequential record within a data block. The KSAM routines which are used for sequential data processing are KPOINTN and KREADN.

The KSAM record management system was designed to meet seven objectives in addition to the basic objectives of a primary key retrieval system. A discussion of how the KSAM record management system meets each of these objectives follows.

The first additional design objective was that the KSAM record management system should have the ability to keep track of more than one data record at a time. This ability is provided by means of the positional pointers. The user specifies, with the call to the KOPEN routine, exactly how many records may be kept track of simultaneously. Each positional pointer may be set to point to a different data record.

The KSAM record management system also had to recover and reuse the space freed by delete operations. As an indivi-

dual data record is deleted from the KSAM file the space that that particular record occupied is recovered simultaneously. The records within the data block are shifted to reuse the space freed and the amount of available space within the block is increased by the amount freed. If enough data records are deleted from a data block eventually that data block will become empty. In this case the empty data block is removed from the KSAM data structure and is added to the list of blocks available for reuse. If enough sons of an index block are freed by delete operations eventually that index block will contain only one index record and thus is no longer necessary. In this case the redundant index block is removed from the KSAM data structure and added to the list of blocks available for reuse. In this manner all space freed by delete operations, both at the record level and at the block level, is recovered and made available for reuse. The KSAM record management system makes no attempt to free up blocks containing only one or two records. This would require a far more sophisticated algorithm than the one employed.

KSAM had to allow for the easy and consistent expansion of the file. The KSAM file may be expanded by a data block split which adds a new data block to the KSAM file. The new data block is added at the point where it is needed not simply at the end of the KSAM file. During the course of a data block split only two information blocks already present in

the KSAM file need to be modified. The father index block and the data block to be split are changed in minor ways. This process may be used to add both data and index blocks to the KSAM data structure. This capability of adding both data and index blocks to the KSAM file provides a consistent and straightforward method of file expansion.

The fourth additional design objective of the KSAM record management system was to allow the KSAM file to be initially loaded with data records sorted, by key, into ascending order. The high value index record is used to direct loading at the end of the file into the last data block in the KSAM data structure. This implies that all records added at the end of the file are added to the end of a data block. The addition of a record at the end of a data block does not require the movement of any of the other data records within the data block. Data records may be added to the KSAM file in ascending order without an excessive amount of work.

The KSAM record management system also had to provide support for both indexed and sequential access to the data records. Indexed access to the data records is provided by the KSAM routines KPOINT and KREADK. The KSAM routines KPOINTN and KREADN are used to provide sequential data processing capability. These access routines may be used in any combination to provide a mixture of indexed data processing and sequential data processing.

The KSAM record management system also had to minimize the number of input and output operations required. The use of the buffering system allows for access to several KSAM information blocks without an input or an output operation being performed. This implies that only blocks which have been changed need to be written onto external storage and only blocks that are required but not already present need to be read. The copy of the block in main memory is used until all of the processing which requires that block has been completed. The number of input and output operations required for each block has been held to a minimum.

The final additional design objective of the KSAM record management system was to provide data integrity. Each of the algorithms that change the structure of the KSAM file was created with this objective in mind. Each algorithm is carefully designed so that a system or power failure at any stage will not result in the loss or change of the data records. In the worst possible situations an empty data or index block may be freed and not added to the free block chain or one or more records may appear more than once if a split operation is interrupted but under no circumstances will data records be inadvertantly lost or changed.

The KSAM record management system requires only a small overhead for the maintenance of control information and indices. Tests have shown that index blocks account for less

than 6% of the total number of blocks in a file, although this depends on the sizes of the keys and the data records.

The KSAM record management system does contain two noticeable areas where the solution to the design objectives may not be the best one. The splitting algorithms tend to produce an unbalanced data structure with more levels of index blocks on the right hand side of the tree. This is caused by the root split algorithm which leaves the first few index records in the root block and moves the upper half of the index records to the new block. This could create problems if all data record searches must take approximately the same amount of time. This problem will be especially acute if the tree is of a very large size. The "point-next" algorithm is a very efficient algorithm except when the next sequential data record is outside of the current data block. In this case the next data record is found using the search algorithm. There will be a large time disparity between these two types of point next operations.

The first of these two problems could be alleviated by a change to the root block split algorithm. This change would result in the creation of a balanced tree at the expense of a more complicated root block split algorithm which would create two new index blocks. The difficulties with the point next algorithm could be reduced with the addition of a brother pointer between data blocks. This, however, can lead

to data record duplication problems and a more complicated data block split algorithm.

The KSAM record management system meets all of the design objectives originally desired. In addition to the basic requirements of a primary key retrieval system the KSAM record management system also meets the other secondary objectives that make the system more convenient and more efficient than a basic system.

BIBLIOGRAPHY

Brillinger,Peter,C. and Cohen,Doren,J.; Introduction to Data Structures and Non-Numeric Computation Englewood Cliffs, N.J.: Prentice Hall, 1972

Ferch,Howard,J.; Using the PDP-11 with UNIX Winnipeg, Man.: The University of Manitoba Dept. of Computer Science,1976

I.B.M.; OS/VS2 Access Method Services San Jose, Calif.: I.B.M. Programming Publishing, 1978

I.B.M.; Enhanced VSAM System Information San Jose, Calif.: I.B.M. Programming Publishing, 1975

I.B.M.; OS/VS1 Virtual Storage Access Method(VSAM) Logic San Jose, Calif.; I.B.M. Programming Publishing, 1975

I.B.M.; OS/VS Virtual Storage Access Method(VSAM) Options for Advanced Applications San Jose, Calif.: I.B.M. Programming Publishing, 1975

I.B.M.; OS/VS Virtual Storage Access Method(VSAM) Programmer's Guide San Jose, Calif.: I.B.M. Programming Publishing, 1978

Knuth,Donald,E.; The Art of Computer Programming Vol.1 Fundamental Algorithms, 2nd Ed. Reading, Mass.: Addison Wesley, 1973

Knuth,Donald,E.; The Art of Computer Programming Vol.3 Searching and Sorting Reading, Mass.: Addison Wesley, 1975

Ritchie,Dennis,M.; C Reference Manual Murray Hill, N.J.: Bell Telephone Laboratories, 1974

Weitzman,Cay; Minicomputer Systems Structure, Implementation, and Application Englewood Cliffs, N.J.: Prentice Hall, 1974