Introduction to
RMS-11

Order No. AA-0001A-TC

digital equipment corporation · maynard. massachusetts

First Printing, January 1977

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECsystem-20	TYPESET-11

CONTENTS

CONTENTS (Cont.)

FIGURES

TABLES

PREFACE

This manual describes the concepts and facilities of Record Management Services for the PDP-11 (RMS-11).

RMS-11 is supported on a number of PDP-11 operating systems. The indexed file organization and associated facilities are extended features available with the RMS-11K product. The COBOL-11, BASIC-PLUS-2, and MACRO-11 languages provide access to RMS-11 features. This manual, therefore, is intended for users with varying operating system and language backgrounds. Since certain capabilities are available only to MACRO-11 programmers, you should consult the reference manual and user's guide associated with a particular language processor for detailed information on the use of RMS-11.

CHAPTER 1

INTRODUCTION

1.1 RMS-11 OVERVIEW

Record Management Services for the PDP-11 (RMS-11) is a set of general-purpose file-handling capabilities. In combination with a host operating system, it provides you with efficient and flexible facilities for data storage, retrieval, and modification. When writing programs in BASIC-PLUS-2, COBOL-11, or MACRO-11, you can select processing methods from among RMS-11's file structuring and accessing techniques that are best suited to a specific application.

To understand how RMS-11 provides this flexibility, you need to be aware of four RMS-11 concepts:

1. File organizations and access modes
2. File attributes
3. Program operations on RMS-11 files
4. The runtime environment

The manner in which RMS-11 builds a file is called its organization. RMS-11 provides three file organizations -- sequential, relative, and indexed. The organization of a file establishes the techniques you can use to retrieve and store data in the file. These techniques are known as access modes. The access modes that RMS-11 supports are 1) sequential, 2) random, and 3) record's file address (RFA).

You can use an application program or an RMS-11 utility, when you create an RMS-11 file, to specify the organization and characteristics of the file. These characteristics are known as the attributes of the file. Among the attributes you specify are storage medium, file and protection specifications, record format and size, and file allocation information.

After RMS-11 creates a file according to the file attributes you specify, application programs can store, retrieve, and modify data in it. These program operations can occur at the logical or physical level. At the logical level, an RMS-11 file is a collection of individual records. The record is the unit of information to which RMS-11 provides access. At the physical level, a file is a collection of units called virtual blocks. When bypassing the record processing capabilities of RMS-11, programs access these virtual blocks through a technique known as block I/O.

During runtime, RMS-11 and the host operating system provide an environment for user programs that permits file sharing and reduces the number of buffers required. When a program accesses files at the logical level, RMS-11 additionally supports 1) multiple access streams

to a single file, 2) synchronous or asynchronous record operations, and 3) move and locate record transfer modes.

1.2 PRESENTATION OF INFORMATION IN THIS MANUAL

The presentation of information in this manual corresponds to these four RMS-11 concepts.

Chapter 2 describes the logical structure, or organization, of RMS-11 files and the access modes available with each file organization.

Chapter 3 details file attributes and their role in creating an RMS-11 file.

Chapter 4 summarizes program operations, at both the logical and physical level, that permit the retrieval, storage, and modification of data in files.

Chapter 5 is an overview of the runtime environment within which user programs process RMS-11 files.

CHAPTER 2

RMS-11 FILE ORGANIZATIONS AND ACCESS MODES

This chapter describes the logical structure and accessing capabilities of RMS-11 files.

A file is a collection of related information. Application requirements establish the nature of this information. For example, a company might maintain personnel information (employee names, addresses, job titles, and so forth) in one file and product information (part numbers, prices, specifications, and so forth) in a second, separate file. Within each of these files, the information is divided into records. In the personnel file, it would be logical for all the information on a single employee to constitute a single record and for the number of records in the file to equal the number of employees. Similarly, each record in the product information file would represent a description of a single product. Again, the number of records in the file reflects the requirements of a particular application -- in this case, a central registry of products sold by a company.

Each record in the personnel and product files would be subdivided into discrete pieces of information known as data fields. The user would define the number, location within the record, and logical interpretation of these data fields. Programmers at the company's data processing installation would write applications that always interpret a particular data field in records of the personnel file as the name of an employee. Likewise, they would interpret another data field, in records of the product file, as a part number. Figure 2-1 illustrates records that might occur in a personnel and a product file.

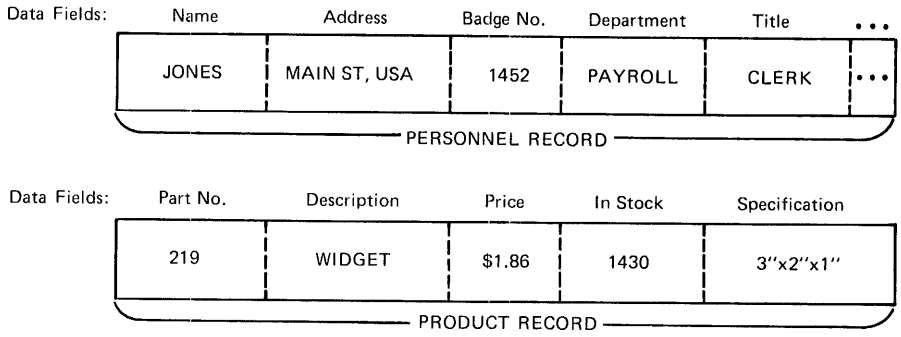

Figure 2-1 Personnel and Product Records

Thus, you can completely control the grouping of data fields into records and records into files. The relationship among data fields and records is known to you and is embedded in the logic of your

programs. In contrast, RMS-11 does not require or employ an awareness of logical relationships among information in the files. Rather, RMS-11 processes records as single units of data. Your programs either build records and pass them to RMS-11 for storage in a file or issue requests for records while RMS-11 performs the necessary operations to retrieve the records from a file.

The purpose of RMS-11, then, is to ensure that every record written into a file can be subsequently retrieved and passed to a requesting program as a single logical unit of data. The structure, or organization, of a file establishes the manner in which RMS-11 stores and retrieves records. The way a program requests the storage or retrieval of records is known as the access mode. The access mode that can be used depends on the organization of a file. The sections of this chapter, therefore, describe:

- RMS-11 file organizations
- RMS-11 access modes
- Sample uses of RMS-11 file organizations.

2.1 RMS-11 FILE ORGANIZATIONS

When creating a file, you have a choice of three file organizations:

1. Sequential
2. Relative
3. Indexed

2.1.1 The Sequential File Organization

In the sequential file organization (see Figure 2-2), records appear in physical sequence. Each record, except the first, has another record preceding it, and each record, except the last, has another record following it. The physical order in which records appear is always identical to the order in which the records were originally written to the file by an application program.

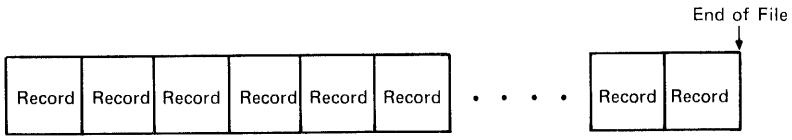

Figure 2-2 Sequential File Organization

2.1.2 The Relative File Organization

When you select the relative organization, RMS-11 structures a file as a series of fixed-size record cells. Cell size is based on the size you specify as the maximum permitted length for a record in the file. RMS-11 considers these cells as successively numbered from 1 (the first) to n (the last). A cell's number represents its location relative to the beginning of the file.

Each cell in a relative file can contain a single record. There is no requirement, however, that every cell contain a record. Empty cells can be interspersed among cells containing records.

Since cell numbers in a relative file are unique, you can use them to identify both a cell and the record, if any, occupying that cell. Thus, record number 1 occupies the first cell in the file, record number 17 occupies the seventeenth cell, and so forth. When you use a cell number to identify a record, it is also known as a relative record number. Figure 2-3 depicts the structure of a relatively organized file.

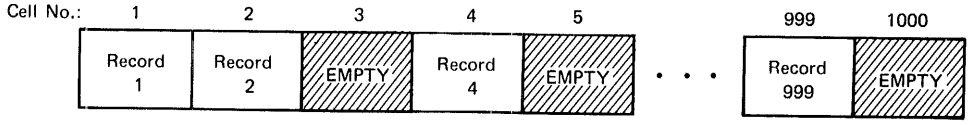

Figure 2-3 Relative File Organization

2.1.3 The Indexed File Organization[1]

Unlike the physical ordering of records in a sequential file or the relative positioning of records in a relative file, the location of records in the indexed file organization is transparent to your program. RMS-11 completely controls the placement of records in an indexed file. The presence of keys in the records of the file governs this placement.

A key is a character string present in every record of an indexed file. The location and length of this character string is identical in all records. When creating an indexed file, you decide which character string in the file's records is to be a key. By selecting such a character string, you indicate to RMS-11 that the contents (i.e., key value) of that string in any particular record written to the file can be used by a program to identify that record for subsequent retrieval.

You must define at least one key for an indexed file. This mandatory key is the primary key of the file. Optionally, you can define additional keys (i.e., alternate keys). Each alternate key represents an additional character string in records of the file. The key value in any one of these additional strings can also be used as a means of identifying the record for retrieval.

As programs write records into an indexed file, RMS-11 locates the values contained in the primary and alternate keys. From the values in keys within records, RMS-11 builds a tree-structured table known as an index. An index consists of a series of entries. Each entry contains a key value copied from a record that a program wrote into the file. With each key value is a pointer to the location in the file of the record from which the value was copied. RMS-11 builds and maintains a separate index for each key you define for the file. Each index is stored in the file. Thus, every indexed file contains at least one index -- the primary key index. When you define alternate keys, RMS-11 builds and stores an additional index for each alternate key. Figure 2-4 shows the general structure of an indexed file that

1. The indexed file organization is supported by the RMS-11K product.

has been defined with only a single key. Figure 2-5 depicts an indexed file defined with two keys -- a primary key and one alternate key.

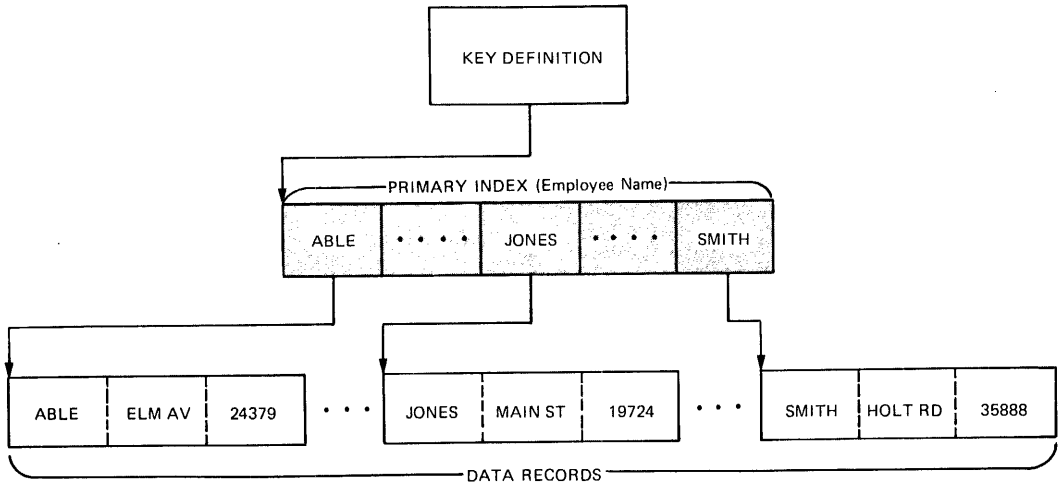

Figure 2-4 Single Key Indexed File Organization

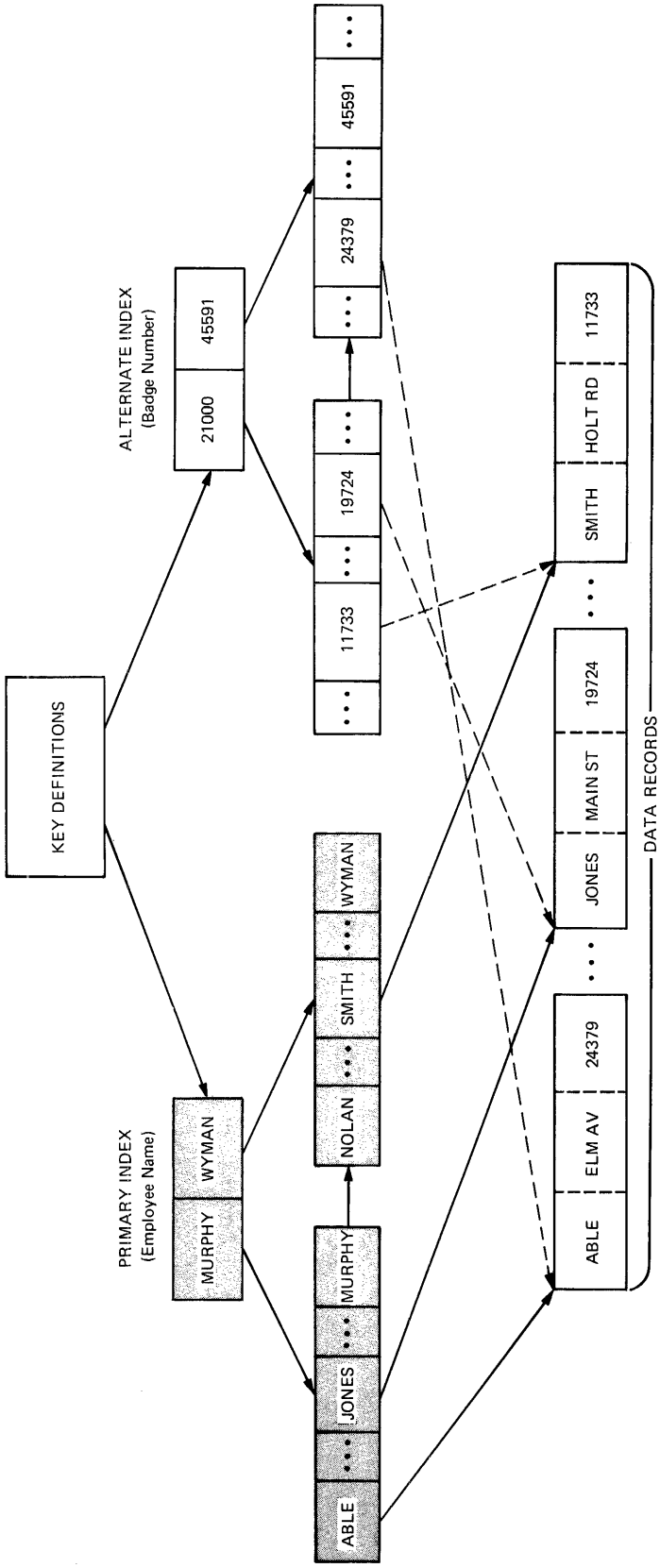

Figure 2-5 Multi-key Indexed File Organization

2.2 RMS-11 ACCESS MODES

The various methods of retrieving and storing records in a file are called access modes. While you must choose the organization of a file at the time you create it, you can use a different access mode to process records within the file each time you open it. Additionally, your program can change access mode during the processing of a file.

RMS-11 provides three record access modes:

1. Sequential
2. Random
3. Record's file address (RFA)

RMS-11 permits only certain combinations of file organization and access mode. Table 2-1 lists these combinations.

Table 2-1
Permissible Combinations of
Access Modes and File Organizations

File Organization	Access Mode			
	Sequential	Random		RFA
		Record #	Key Value	
Sequential	Yes	No	No	Yes[1]
Relative	Yes	Yes	No	Yes
Indexed	Yes	No	Yes	Yes

[1]Disk files only.

The following subsections describe RMS-11 access modes and the capability of changing access mode during program execution.

2.2.1 Sequential Access Mode

You can use sequential access mode to access all RMS-11 files. Sequential access means that records are retrieved or written in a particular sequence. The organization of the file established this sequence.

2.2.1.1 **Sequential Access to Sequential Files** - In a sequentially organized file, physical adjacency establishes the order in which records are retrieved when you use the sequential access mode. To read a particular record in a file, say the fifteenth record, a program must open the file and access the first fourteen records before accessing the desired record. Thus, each record in a sequential file can be retrieved only by first accessing all records that physically precede it. Similarly, once a program has retrieved the fifteenth record, it can read all the remaining records (from the

sixteenth on) in physical sequence. It cannot, however, read any preceding record without closing and reopening the file and beginning again with the first record.

When writing new records to a sequential file in sequential access mode, a program must first request that RMS-11 position the file immediately following the last record. Thereafter, each sequential write operation the program issues causes a record to be written following the previous record.

2.2.1.2 **Sequential Access to Relative Files** - During the sequential access of records in the relative file organization, the contents of the record cells in the file establish the order in which a program processes records. RMS-11 has the ability to recognize whether successively numbered record cells are empty or contain records.

When a program issues read requests in sequential access mode for a relative file, RMS-11 ignores empty record cells and searches successive cells for the first one containing a record. If, for example, a relative file contains records only in cells 3, 13, and 47, successive sequential read requests cause RMS-11 to return relative record number 3, then relative record number 13, and finally relative record number 47.

When a program adds new records in sequential access mode to a relative file, the order in which RMS-11 writes the records depends on ascending relative cell numbers. Each write request causes RMS-11 to place a record in the cell whose relative number is one higher than the relative number of the previous request -- as long as that cell does not already contain a record. If the cell already contains a record, RMS-11 rejects the write operation. Thus, RMS-11 allows a program to write new records only into empty cells in the file.

2.2.1.3 **Sequential Access to Indexed Files** - In an indexed file, the presence of one or more indexes permits RMS-11 to determine the order in which to process records in sequential access mode. The entries in an index are arranged in ascending order by key values. Thus, an index represents a logical ordering of the records in the file. If you have defined more than one key for the file, each separate index associated with a key represents a different logical ordering of the records in the file. A program, then, can use the sequential access mode to retrieve records in the order represented by any index.

When reading records in sequential access mode from an indexed file, a program initially specifies a key (e.g., primary key, first alternate key, second alternate key, etc.) to RMS-11. Thereafter, RMS-11 uses the index associated with that specified key to retrieve records in the sequence represented by the entries in the index. Each successive record RMS-11 returns in response to a program read request contains a value in the specified key field that is equal to or greater than that of the previous record returned.

In contrast to a sequential read request, sequential write requests to an indexed file do not require the initial key specification. Rather, RMS-11 uses the stored definition of the primary key field to locate the primary key value in each record to be written to the file. When a program issues a series of sequential write requests, RMS-11 verifies that each successive record contains a key value in the primary key field that is equal to or greater than that of the preceding record.

2.2.2 Random Access Mode

In random access mode, the program, rather than the organization of the file, establishes the order in which records are processed. Each program request for access to a record operates independently of the previous record accessed. Associated with each request in random mode is an identification of the particular record of interest. Successive requests in random mode can identify and access records anywhere in the file.

You cannot use random access mode with sequentially organized files. Both the relative and indexed file organizations, however, permit random access to records. The subsections that follow describe the use of random access with these organizations. Each organization provides a distinct way programs can identify records for access.

2.2.2.1 **Random Access in Relative Files** - Programs can read or write records in a relative file by specifying relative record numbers. RMS-11 interprets each such number as the corresponding cell in the file. A program, therefore, can read records at random by successively requesting, for example, record number 47, record number 11, record number 31, and so forth. If no record exists in a specified cell, RMS-11 returns a nonexistence indicator to the requesting program. Similarly, a program can store records in a relative file by identifying the cell in the file that a record is to occupy. If a program attempts to write a new record in a cell already containing a record, RMS-11 returns a record-already-exists indicator to the program.

2.2.2.2 **Random Access to Indexed Files** - The indexed file organization also permits random access of records. However, for indexed files, a key value, rather than a relative record number, identifies the record.

Each program read request in random access mode specifies a key value and the index (e.g., primary index, first alternate index, second alternate index, etc.) RMS-11 must search. When RMS-11 finds the key value in the specified index, it reads the record the index entry points to and passes the record to the user program.

In contrast to read requests, which require a program-specified key value, program requests to write records randomly in an indexed file do not require the separate specification of a key value. All key values (primary and, if any, alternate key values) are in the record itself. When you open an indexed file, RMS-11 retrieves all key definitions stored in the file. Thus, RMS-11 knows the location and length of each key field in a record. Before writing a record into the file, RMS-11 examines the values contained in the key fields and creates new entries in the indexes. In this way, RMS-11 ensures that the record can be retrieved by any of its key values. Thus, the process by which RMS-11 adds new records to the file is precisely the process it uses to construct the original index or indexes.

2.2.3 Record's File Address (RFA) Access Mode

You can use record's file address (RFA) access mode with any file organization as long as the file resides on a disk device. This access mode is further limited to retrieval operations only. Similar

to random access mode, however, RFA access allows a specific record to be identified for retrieval.

As the term record's file address indicates, every record within a file has a unique address. The actual format of this address depends on the organization of the file. In all instances, however, only RMS-11 can interpret this format.

The most important feature of RFA access is that the address (RFA) of any record remains constant while the record exists in the file. After every successful read or write operation, RMS-11 returns the RFA of the subject record to your program. Your program can then save this RFA to use again to retrieve the same record. It is not required that this RFA be used only during the current execution of your program. RFAs can be saved and used at any subsequent point in time.

2.2.4 Dynamic Access

Dynamic access is not strictly an access mode. Rather, it is the capability to switch from one access mode to another while processing a file. There is no limitation on the number of times such switching can occur. The only limitation is that the file organization (or, in the case of RFA access, the device containing the file) must support the access mode selected.

As an example, you can use dynamic access effectively immediately following a random or RFA access mode operation. When your program accesses a record in one of these modes, RMS-11 establishes a new current position in the file. Your program can then switch to sequential access mode. By using the randomly accessed record (rather than the beginning of the file) as the starting point, your program can retrieve succeeding records in the sequence established by the file's organization.

2.3 SAMPLE USES OF RMS-11 FILE ORGANIZATIONS

You choose a file organization according to your application's need to access records in a particular way. The following subsections suggest three situations in which a needed access mode determines the selection of the sequential, relative, or indexed file organization.

2.3.1 Sample Sequential File

Consider a particular requirement of a mail-order company as an example of the use of the sequential file organization. Such a company might maintain its master list of customer names and addresses as individual records in a sequential file. This means of organizing information would be efficient for a mass mailing application. A program would access each record in the file in turn and print its contents on a separate mailing label. The program would read the entire file since each record is of equal interest. However, the same file organization would not be efficient if the program needed to access the address of a single individual. In this case, each individual record would have to be read and examined until the correct one was found.

2.3.2 Sample Relative File

A company might choose to maintain product information records in a relative file. The file would be organized so that each cell number (and, therefore, relative record number) would correspond with the unique part number associated with a particular product. This structure would allow easy access to any individual product record in the file if the part number associated with that product were known. Unlike records in a sequential file, there would be no need to access any record in the relative file other than the record of interest.

2.3.3 Sample Indexed File

In creating a file to contain personnel information, a company might select the indexed organization. The employee badge number field might be selected as the primary key. This field, rather than the employee name field, is a suitable primary key because badge numbers within a company are unique, whereas two employees could have the same name. Access to the personnel record of any individual, then, would require only the badge number. The record RMS-11 would return to the application program would contain the specified value in the badge number field. As in the relative file organization, access to a particular record in an indexed file does not require that any other records be accessed first.

CHAPTER 3

FILE ATTRIBUTES

You can use either an application program (written in COBOL-11, BASIC-PLUS-2, or MACRO-11) or the RMS-11 utility called DEFINE (refer to the RMS-11 Utilities User's Guide) to create a file. Each RMS-11 file has certain logical and physical characteristics, known as attributes. Through the source language statements of an application program or the command line invoking the DEFINE utility, you describe these attributes to RMS-11. With this attribute information, RMS-11 begins the structuring of the file on a storage medium.

The most important attribute of any RMS-11 file is its organization. You can tailor a file for use in a particular application by making the proper selection of this and other required or optional attributes. In addition to file organization, you can choose from among the following attributes:

- The storage medium on which the file resides
- The file and protection specification of the file
- The format and size of records
- The size of the file
- The size of a particular storage structure, known as the bucket, within relative and indexed files
- The definition of keys for indexed files.

3.1 STORAGE MEDIA

Your selection of a storage medium on which RMS-11 builds a file is related to the organization of the file. You can create permanent sequential files on disk devices or ANSI magnetic tape volumes. Transient files can be written on devices such as line printers and terminals.

Unlike sequential files, relative and indexed files are restricted to a particular medium. These files can reside only on disk devices.

3.2 FILE AND PROTECTION SPECIFICATIONS

The name you assign to a new file enables RMS-11 to subsequently find the file on the storage medium. You follow the conventions for file specifications of the host operating system. Such conventions may differ among the systems supporting RMS-11.

RMS-11 also allows you to assign a protection specification to a file at the time you create it; again, the format of this specification is the format used by the host operating system. Potential users of the file are classified according to membership in the following classes:

1. System - privileged users as determined by a particular installation (This class is not recognized on RSTS/E systems, where privileged users have unlimited access to files.)
2. Owner - the account under which the file is created
3. Group - users with the same project identifier as the owner
4. World - users in general.

Within each class, you may explicitly elect to deny any or all of the following types of access:

- Read access
- Write access
- Other types of access as determined by the host operating system.

Exact descriptions of membership in class and types of access supported are described in manuals pertaining to the host operating system.

3.3 FORMAT AND SIZE OF RECORDS

When creating a file, you must provide format and maximum size specifications for the records the file will contain. The specified format establishes how each record physically appears in the file on a storage medium. The size specification allows RMS-11 to subsequently verify that records written into the file do not exceed the length specified when you created the file.

3.3.1 RMS-11 Record Formats

RMS-11 supports four record formats:

1. Fixed
2. Variable
3. Variable-with-fixed-control (VFC)
4. Stream

Similar to the selection of a storage medium, the choice of a format for the records of a file depends on a file's organization. Table 3-1 shows the allowed combinations of record format and file organization.

Table 3-1
Record Formats and File Organizations

File Organization	Record Format			
	Fixed	Variable	VFC	Stream
Sequential	Yes	Yes	Yes	disk only
Relative	Yes	Yes	Yes	No
Indexed	Yes	Yes	No	No

3.3.1.1 **Fixed Length Record Format** - The term fixed length record format refers to records of a file that are all equal in size. Each record, then, occupies an identical amount of space in the file.

3.3.1.2 **Variable-Length Record Format** - In variable-length record format, records in a file can be either equal or unequal in length. To allow retrieval of variable-length records from a file, RMS-11 prefixes a count field to each record it writes. The count field describes the length (in bytes) of the record. RMS-11 removes this count field before it passes a record to your program.

RMS-11 produces two types of count fields, depending on the storage medium on which the file resides:

1. Variable-length records in files on disk devices have a 1-word (2-byte) binary count field preceding the data field portion of each record. The specified size excludes the count field.

2. Variable-length records on ANSI magnetic tapes have 4-character decimal count fields preceding the data portion of each record. The specified size includes the count field. In the context of ANSI tapes, this record format is known as D format.

3.3.1.3 **Variable-with-Fixed-Control Record Format** - From your point of view, variable-with-fixed-control (VFC) records consist of two distinct parts, the fixed control area and the user data record. The size of the fixed control area is identical for all records of the file. The contents of each fixed control area are completely under the control of your program and can be used for any purpose. As an example, you might use fixed control areas to store the identifier (e.g., relative record number or RFA) of related records.

The second part of a VFC record is similar to a variable-length record. In other words, it is a user data record, variable in length, and composed of individual data fields.

The two parts of a VFC record correspond to the way your program writes and reads such records. Prior to an output operation, your program builds a VFC record in two locations. It builds the fixed control area in a location separate from the user data part of the record. When writing the record to the file, RMS-11 fetches both the

fixed control area and the user data part of the record from their respective program locations. RMS-11 then prefixes the user data part of the record with the fixed control area, prefixes the result with a count field that describes the total size of both parts, and writes the record to the file.

On input operations, RMS-11 reverses the preceding procedure. It uses the count field to locate the entire VFC record in the file. RMS-11 removes this count field. Then, it removes the fixed control area from the record and stores it in one program location while storing the remaining part in a second location.

3.3.1.4 **Stream Format Records** - Records in stream format can be variable in size. However, no count field precedes each record. Instead, RMS-11 considers the entire file a stream of contiguous ASCII characters. Each record in the file is delimited by one of the following:

1. Form feed (FF)
2. Vertical tab (VT)
3. Line feed (LF)
4. Carriage return immediately followed by line feed (CR-LF)

NOTE

Stream format records are supported for file interchange with non-RMS-11 application programs. Since this format is highly inefficient, it should be used only when such interchange is a concern.

On output operations, RMS-11 examines the last character of the record constructed by a program. If this character is an LF, VT, or FF, RMS-11 leaves the record unaltered and writes it to the file. If the last character is not LF, VT or FF, RMS-11 appends a carriage return (CR) character followed by a line feed (LF) character to the record before writing it to the file.

On input operations, RMS-11 scans the stream of ASCII characters, removing NUL characters and searching for the first occurrence of an FF, VT, LF, or CR-LF combination. If the character that terminates the scan is an FF, VT, or LF (not preceded by CR), RMS-11 passes the entire string, including the terminating character, to the program. If, however, the scan encounters a CR-LF combination, RMS-11 removes these two characters and passes the preceding string as a record to the program. Each successive input operation causes the scan to resume at the character following the last FF, VT, LF, or CR-LF combination encountered.

3.3.2 Size of Records

You must provide RMS-11 with record size information along with the selected record format. RMS-11's use of this information depends on the record format chosen.

When you choose fixed format records, you must indicate the actual size of each record in the file. This size specification becomes part of the information stored and maintained by RMS-11 for the file. Thereafter, if a program attempts to write a record whose length differs from this specified size, RMS-11 will reject the operation.

When creating a file with variable-length format records, you can specify a maximum record size greater than zero or, for sequential and indexed files only, a maximum record size equal to zero. If the specified size is greater than zero, RMS-11 interprets the value as the size of the largest record that can be written into the file. Subsequently, each record that a program actually writes must be less than or equal to this value or RMS-11 rejects the operation. When you specify a record size of zero for a sequential or indexed file, RMS-11 neither checks nor enforces a maximum record size.

VFC format records require two size specifications. The first size specification identifies the length of the fixed control area of all records in the file. The second size specification represents the maximum length of the data portion of the VFC records. RMS-11 handles this second size specification in a manner similar to its handling of the size specification for variable format records. RMS-11 interprets a nonzero value as the size of the largest data portion of a VFC record that a program can write into the file. When, however, this second size specification is zero, RMS-11 neither checks nor enforces a maximum record size.

Finally, for stream format records, RMS-11 permits you to specify the same record size information as for variable format records. That is, a nonzero value represents the maximum permitted size of any record written in the file while a zero value suppresses RMS-11's size checking.

3.4 SIZE OF RMS-11 FILES

The size of an RMS-11 file is expressed as an integral number of virtual blocks. Virtual blocks are physical storage structures. That is, each virtual block in a file is a unit of data whose size depends on the physical medium on which the file resides. For example, the size of virtual blocks in files on disk devices is 512 bytes. Operating system convention has established this size; you cannot alter it. On magnetic tape, a virtual block is the information written between two interrecord gaps. When using ANSI-labeled magnetic tapes, you can specify the size of virtual blocks within files.

The operating system assigns ascending numbers to a file's virtual blocks. This numbering scheme allows a file to appear to you, and to RMS-11 itself, as a series of adjacent virtual blocks. In reality, however, the successive numbering of virtual blocks and the physical placement of these blocks on a storage medium need not necessarily correspond. For example, virtual blocks 167 and 168 in a file, although adjacent in terms of the numbering scheme applied to virtual blocks, need not necessarily occupy adjacent physical locations. Once again, the physical medium on which the file resides determines the handling of virtual blocks.

On magnetic tapes, successively numbered virtual blocks actually occupy successive physical locations. Virtual blocks from one file are never intermixed with virtual blocks from another. On disk devices, however, the situation can be quite different. Files on disk can reside in one or more discrete areas known as extents (see Figure

3-1). Within a single file extent, successively numbered virtual blocks occupy successive physical locations. If a file consists of more than one extent, however, extents of other files or unused space can be interspersed among the extents of the first file.

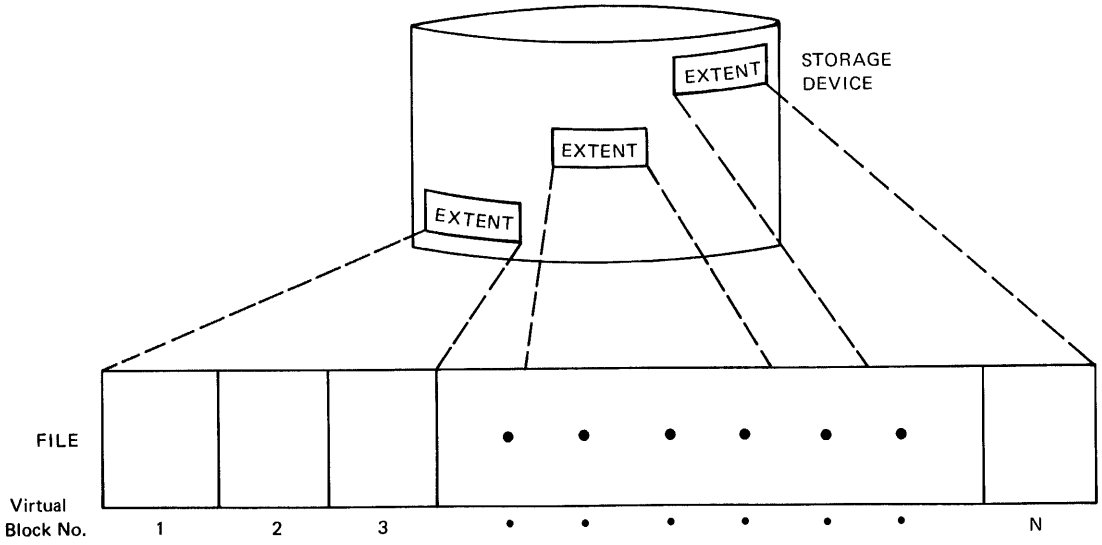

Figure 3-1 Virtual Blocks and Extents

When a file consists of only one extent, it is called a contiguous file since all successive virtual blocks of the file correspond to successive contiguous physical locations. When creating a file, you can request that RMS-11 allocate the file contiguously. The default action is that RMS-11 allocates contiguously as much of the file as possible.

The virtual blocks of a file contain the records your programs write into the file. Depending on the size of records, a virtual block can contain one record, more than one record, or a portion of a record. When a particular record is larger than the size of a virtual block, it is stored in as many successively numbered virtual blocks as needed to contain the entire record. In this instance, the record is said to cross or span block boundaries.

When creating an RMS-11 file, you can specify an initial allocation size. If no file size information is given, RMS-11 allocates the minimum amount of storage needed to contain the defined attributes of the file. If you do specify an initial allocation size, RMS-11 obtains the specified number of virtual blocks for the file. However, this initial size specification does not represent a permanent limit on the size of the file. RMS-11 automatically extends any file whenever the presently allocated space is insufficient to satisfy a program request for storage of data.

3.5 BUCKETS IN RELATIVE AND INDEXED FILES

RMS-11 uses a storage structure known as a bucket for building and maintaining relative and indexed files. Unlike virtual blocks, a bucket can never contain a portion of a record. That is, RMS-11 never permits records to span bucket boundaries.

You define the size of buckets in a file at the time you create the file. Buckets can consist of from 1 to 32[1] virtual blocks. When selecting a bucket size, you must consider file organization, record format, record size, and the internal information RMS-11 maintains in each bucket. Within these constraints, a large bucket size will serve to increase sequential mode processing of a file since fewer actual I/O transfers are required to access records. Minimizing bucket size, on the other hand, means that less I/O buffer space is required to support file processing.

3.6 KEY DEFINITIONS FOR INDEXED FILES

To define a key for an indexed file, you must specify the position and length of character data in the records of the file. You must define at least one key--the primary key--for an indexed file. Additionally, up to 254 alternate keys can be defined. Each such key, primary and alternate, represents from 1 to 255 characters in each record of the file.

When identifying the position and the length of keys to RMS-11, you can define simple or segmented keys. A simple key is a single, contiguous string of characters in the record. In other words, it is a single data field. A segmented key, however, can consist of from two to eight data fields within records. These data fields need not be contiguous. When processing records that contain segmented keys, RMS-11 treats the separate data fields (segments) as a logically contiguous character string.

When defining keys at file creation time, you can specify two characteristics for each key:

1. Duplicate key values are allowed.

2. Key value can change.

When you specify that duplicate key values are allowed, you indicate that more than one record in the file can have the same value in a given key. Such records, therefore, have the same record identifier. The capability to allow duplicate key values further distinguishes indexed files from relative files. In relative files, the record identifier, representing a relative record number, is always unique.

The personnel file can serve as an example of the use of duplicate keys. At file creation time, the creator of the file could define the department name field as an alternate key. As programs wrote records into the file, the alternate index for the department name key field would contain multiple entries for each key value (e.g., PAYROLL, SALES, ADMINISTRATION) since departments are composed of more than one employee. When such duplication occurs, RMS-11 stores the records so that they can be retrieved in first-in/first-out (FIFO) order.

Using the preceding personnel file, an application could be written to list the names of employees in any particular department. A single execution of the application could list the names of all employees working, for example, in the department called SALES. By randomly accessing the file by alternate key and the key value SALES, the application would obtain the first record written into the file containing this value. Then, the application could switch to

---

1. The maximum bucket size on the RSTS/E operating system is 15 virtual blocks.

sequential access and successively obtain records with the same value, SALES, in the alternate key field. Part of the logic of the application would be to determine the point at which a sequentially accessed record no longer contained the value SALES in the alternate key field. The program could then switch back to random access mode and access the first record containing a different value (e.g., PAYROLL) in the department name key field.

The second key characteristic (key value can change) indicates that records can be read and then written back into the file with a modified value in the key. When such modification occurs, RMS-11 automatically updates the appropriate index to reflect the new key value. You can specify this characteristic only for alternate keys. Further, when specifying this characteristic, you must also specify that duplicate key values are allowed.

If the sample personnel file were created with the department name field as an alternate key, the creator of the file would need to specify that key values can change. This specification would allow a program to access a record in the file and change the contents of a department name data field to reflect the transfer of an employee from one department to another.

You can also declare the converse of either of these two key characteristics. That is, you can specify for a given key that duplicate key values are not allowed or that key values cannot change. When duplicate key values are not allowed, RMS-11 rejects any program request to write a record containing a value in the key that is already present in another record. Similarly, when the key value cannot change, RMS-11 does not allow a program to write a record back into the file with a modified value in the key.

CHAPTER 4

PROGRAM OPERATIONS ON RMS-11 FILES

After RMS-11 creates a file according to your description of file attributes, your program can access the file and store and retrieve data. When your program accesses the file as a logical structure (i.e., a sequential, relative, or indexed file), it uses access modes to perform record operations that add, retrieve, update, and delete records. The organization of the file determines the types of record operations permitted. If you choose to bypass the record accessing capabilities of RMS-11, your program can access the file as a physical structure. As a physical structure, RMS-11 considers the file simply an array of virtual blocks. To process a file at the physical level, programs use a type of access known as block I/O.

The two sections that follow describe, respectively, record operations and block I/O.

4.1 RECORD OPERATIONS ON RMS-11 FILES

The organization of a file, defined when you create the file, determines the types of operations that your program can perform on records. Depending on file organization, RMS-11 permits your program to perform the following record operations:

- Read a record. RMS-11 returns an existing record within the file to your program.
- Write a record. RMS-11 adds a new record that your program constructs to the file. The new record cannot replace an already existing record.
- Find a record. RMS-11 locates an existing record in the file. It does not return the record to your program, but establishes a new current position in the file.
- Delete a record. RMS-11 removes an existing record from the file.
- Update a record. Your program modifies the contents of a record read from the file. RMS-11 writes the modified record into the file, replacing the old record.

Table 4-1 shows the cominbations of record operations and file organizations that RMS-11 permits. The subsections that follow discuss record operations in the context of each file organization.

Table 4-1
Record Operations and File Organizations

File Organization	Record Operation				
	Read	Write	Find	Delete	Update
Sequential	Yes	Yes	Yes	No	Yes[1]
Relative	Yes	Yes	Yes	Yes	Yes
Indexed	Yes	Yes	Yes	Yes	Yes

[1]Disk files only.

4.1.1 Sequential File Organization Record Operations

In the sequential file organization, your program can read existing records from the file using sequential or RFA access modes. New records can be added only to the end of the file and only through the use of sequential access mode. The find operation is supported in both sequential and RFA access mode. In sequential access mode your program can use a find operation to skip records. In RFA access mode, your program can use the find operation to establish a random starting point in the file for sequential read operations. The sequential file organization does not support the delete operation since the structure of the file requires that records be adjacent in and across virtual blocks. Your program can, however, update existing records in disk files as long as the modification of a record does not alter its size.

4.1.2 Relative File Organization Record Operations

The relative file organization permits your program greater flexibility in performing record operations than the sequential organization. Your program can read existing records from the file using sequential, random, or RFA access mode. New records can be sequentially or randomly written so long as the intended record cell does not already contain a record. Similarly, any access mode can be used to perform a find operation. After a record has been found or read, RMS-11 permits the delete operation. Once a record has been deleted, the record cell is available for a new record. Lastly, your program can update records in the file. If the format of the records is variable, update operations can modify record length up to the maximum size specified when the file was created.

4.1.3 Indexed File Organization Record Operations

The indexed file organization provides the greatest flexibility in performing record operations. Your program can read existing records from the file in sequential, RFA, or random access mode. When reading records in random access mode, your program can choose one of four

types of matches that RMS-11 must perform using the program-provided key value. The four types of matches are:

1. Exact key match
2. Approximate key match
3. Generic key match
4. Approximate and generic key match

Exact key match requires that the contents of the key in the record retrieved precisely match the key value specified in the program read operation.

The approximate match facility allows your program to select either of the following two relationships between the key of the record retrieved and the key value specified by your program:

- Equal to or greater than
- Greater than

The advantage of this kind of match is that if the requested key value does not exist in any record of the file, RMS-11 returns the record that contains the next higher key value. This allows your program to retrieve records without knowing an exact key value.

Generic key match means that the program need specify only an initial portion of the key value. RMS-11 returns to your program the first occurrence of a record whose key contains a value beginning with those characters. This capability is useful in applications where a series of records must be retrieved according to the contents of only a part of the key field. In an indexed inventory file, for example, a company might designate its part numbers in such a way that the first three digits represent the vendor from whom the part is purchased. In order to retrieve the record associated with a particular part, the program would normally supply the entire part number. Generic selection permits the retrieval of the first record representing parts purchased from a specific vendor.

The final type of key match combines both the generic and approximate facilities. Your program specifies only an initial portion of the key value, as with generic match. Additionally, your program specifies that the key data field of the record retrieved must be either:

- Equal to or greater than the program-supplied value
- Greater than the program-supplied value

In addition to versatile read operations, RMS-11 allows any number of new records to be written into an indexed file. It rejects a write operation only if the value contained in a key of the record violates a user-defined key characteristic (e.g., duplicate key values not permitted).

The find operation, similar to the read operation, can be performed in sequential, RFA, or random access mode. When finding records in random access mode, your program can specify any one of the four types of key matches provided for read operations.

In addition to read, write, and find operations, your program can delete any record in an indexed file and update any record. The only restriction RMS-11 applies during an update operation is that the

contents of the modified record must not violate any user-defined key characteristic (e.g., key values cannot change or duplicate key values not permitted).

4.2 BLOCK I/O

Block I/O allows your program to bypass entirely the record processing capabilities of RMS-11. Rather than performing record operations through the use of supported access modes, your program can process a file as a physical structure consisting solely of virtual blocks.

Using block I/O, your program reads or writes multiple virtual blocks by identifying a starting virtual block number in the file. Regardless of the organization of the file, RMS-11 accesses the identified block or blocks on behalf of your program.

Since RMS-11 files, particularly relative and indexed files, contain internal information meaningful only to RMS-11 itself, DIGITAL does not recommend that you modify a file using block I/O. The presence of the block I/O facility, however, does permit you to create your own file structures. You must, however, maintain the resultant structures using specialized programs. You cannot access these structures through the use of RMS-11 record access modes and record operations.

CHAPTER 5

THE RMS-11 RUNTIME ENVIRONMENT

The environment within which your program processes RMS-11 files at runtime consists of two levels, the file processing level and the record processing level. At the file processing level, RMS-11 and the host operating system provide an environment that permits concurrently executing programs to share access to the same file. RMS-11 ascertains the amount of sharing permissible from information provided by the programs themselves. Additionally, at the file processing level, RMS-11 provides facilities that allow programs to minimize buffer space requirements for file processing. At the record processing level, RMS-11 allows your program to access records in a file through one or more record access streams. Each record access stream represents an independent and simultaneously active series of record operations directed toward the file. Within each stream, your program can perform record operations synchronously or asynchronously. That is, RMS-11 allows your program to choose between receiving control only after a record operation request has been satisfied (synchronous operation) or receiving control possibly before the request has been satisfied (asynchronous operation). Lastly, for both synchronous and asynchronous record operations, RMS-11 provides two record transfer modes, move mode and locate mode. Move mode causes RMS-11 to copy a record from an I/O buffer into a program-provided location. Locate mode allows your program to address records directly in an I/O buffer.

The two sections of this chapter describe, respectively, the file processing and record processing runtime environment.

5.1 THE FILE PROCESSING ENVIRONMENT

RMS-11 provides two major facilities at the file processing level, file sharing and buffer handling.

5.1.1 File Sharing

Timely access to critical files requires that more than one concurrently executing program be allowed to process the same file at the same time. Therefore, RMS-11 allows executing programs to share files rather than requiring them to process files serially. The manner in which a file can be shared depends on the organization of the file. Program provided information further establishes the degree of sharing of a particular file. RMS-11 coordinates the sharing of a

relative or indexed file through a bucket locking mechanism. The following paragraphs, therefore, describe:

- File organizations and file sharing
- Program sharing information
- Bucket locking

5.1.1.1 **File Organizations and File Sharing** - With the exception of magnetic tape files, which cannot be shared, every RMS-11 file can be shared by any number of programs that are reading, but not writing, the file. Sequential files on disk can be shared by multiple readers and a single writer. Relative and indexed files, however, can be shared by multiple readers and multiple writers. Your program can read or write records in a relative or indexed file while other programs are similarly reading or writing records in the file. Thus, the information in such files can be changing while programs are accessing them.

5.1.1.2 **Program Sharing Information** - While a file's organization establishes whether it can be shared for reading with a single writer or for multiple readers and writers, your program specifies whether such sharing actually occurs at runtime. You control the sharing of a file through information your program provides RMS-11 when it opens the file. First, your program must declare what operations (e.g., read, write, delete, update) it intends to perform on the file. Second, your program must specify whether other programs can read the file or both read and write the file concurrently with your program.

The combination of these two types of information allows RMS-11 to determine if multiple user programs can access a file at the same time. Whenever your program's sharing information is compatible with the corresponding information another program provides, both programs can access the file concurrently.

5.1.1.3 **Bucket Locking** - RMS-11 uses a bucket locking facility to control operations to a relative or indexed file that is being accessed by one or more writers. The purpose of this facility is to ensure that a program can add, delete, or modify a record in a file without another program simultaneously accessing the same record.

When your program opens an indexed or relative file with the declared intention of writing or updating records, RMS-11 locks any bucket accessed by your program. This locking prevents another program from accessing any record in the bucket until your program releases it. The lock remains in effect until your program accesses another bucket. RMS-11 then unlocks the first bucket and locks the second. The first bucket is then available for access by another concurrently executing program.

5.1.2 Buffer Handling

To your program, record processing under RMS-11 appears as the movement of records directly between a file and the program itself. Transparently to your program, however, RMS-11 reads or writes virtual

blocks or buckets of a file into or from internal memory areas known as I/O buffers. Records within these buffers are then made available to your program.

The storage structures transferred between a file and I/O buffers depend on the organization of the file. When your program processes sequential files, RMS-11 reads and writes virtual blocks. For relative and indexed files, RMS-11 reads and writes buckets. Thus, the storage element RMS-11 uses to structure the file is the unit of transfer between the file and memory when RMS-11 accesses the file in response to your program's record operation request.

In addition to buffers that contain virtual blocks or buckets, RMS-11 requires a set of internal control structures to support file processing. The combination of these buffers and control structures is known as the space pool. RMS-11 maintains a separate space pool for each executing program. Rather than allocating space solely on the basis of the total number of files processed, RMS-11 provides facilities to ensure that a space pool is large enough to accommodate only the requirements of the largest number of files that can be open simultaneously. Using these facilities, your program provides information that allows RMS-11 to calculate the minimum size requirements of the space pool.

In providing size requirements for the I/O buffer portion of the space pool, you can choose one of two options:

1. A completely centralized space pool

2. Private I/O buffers for one or more files

In a completely centralized space pool, all I/O buffers as well as the internal control structures required for file processing are inaccessible to your program. RMS-11 totally manages the space within the pool and allocates portions, as needed, as buffer space and control structures for open files.

Unlike a completely centralized space pool, private I/O buffers allow your program some measure of control over I/O buffer space. You can allocate private I/O buffers on a per-file basis by explicitly specifying the address and total size of the buffers to be used for a particular file. While the file is open, RMS-11 manages this buffer space and your program must not access it. However, when the file is closed, the private I/O buffer space is available for use by your program.

The major advantage of private I/O buffers is that they avoid fragmenting a completely centralized space pool. That is, since particular files have varying buffer requirements based on their organization, a centralized space pool could have sufficient space available for the opening of an additional file but the space could be noncontiguous. When such a situation arises, your program can not open the desired file. Such fragmentation cannot occur in a private I/O buffer pool since there is no mixture of differing space requirements.

5.2 THE RECORD PROCESSING ENVIRONMENT

After opening a file, your program can access records in the file through the RMS-11 record processing environment. This environment provides three facilities:

1. Record access streams
2. Synchronous or asynchronous record operations
3. Move or locate record transfer modes

5.2.1 Record Access Streams

In the record processing environment, your program accesses records in a file through a record access stream. A record access stream is a serial sequence of record operation requests. For example, your program can issue a read request for a particular record, receive the record from RMS-11, modify the contents of the record, and then issue an update request that causes RMS-11 to write the record back into the file. The sequence of read and update record operation requests can then be performed for a different record, or other record operations can be performed -- again in a serial fashion. Thus, within a record access stream, there is at most one record being processed at any point in time. However, for relative and indexed files, RMS-11 permits your program to establish multiple record access streams for record operations to the same file. The presence of such multiple record access streams allows your program to process in parallel more than one record of a file. Each stream represents an independent and concurrently active sequence of record operations. Further, when such streams update records in the file, RMS-11 employs the same bucket locking mechanism among streams that it uses to control the sharing of a file among separate programs.

As an example of multiple record access streams, your program could open an indexed file and establish two record access streams to the file. Your program could use one record access stream to access records in the file in random access mode through the primary index. At the same time, your program could use the second record access stream to access records sequentially in the order specified by an alternate index. When your program accesses a record through either stream, RMS-11 automatically uses its bucket locking mechanism to ensure that both streams do not attempt to write the same record at the same time.

5.2.2 Synchronous and Asynchronous Record Operations[1]

Within each record access stream, your program can perform any record operation either synchronously or asynchronously. When a record operation is performed synchronously, RMS-11 returns control to your program only after the record operation request has been satisfied (e.g., a record has been read and passed to your program). When a record operation is performed asynchronously, RMS-11 can return control to your program before the record operation request has been satisfied. Your program, then, can utilize the time required for the physical transfer between the file and memory of the block or bucket containing the record to perform other computations. However, your program cannot issue a second record operation through the same stream until the first record operation has completed. To ascertain when a record operation has actually been performed, your program can issue a wait request and regain control when the record operation is complete.

---

1. The RSTS/E operating system supports synchronous record operations only.

5.2.3 Record Transfer Modes

In addition to specifying synchronous or asynchronous operations for each request in a record access stream, your program can utilize either of two record transfer modes to gain access to each record in memory. The following subsections describe these two modes, move mode and locate mode.

5.2.3.1 **Move Mode Record Transfers** - RMS-11 permits move mode record operations for all file organizations and record operations. Move mode requires that an individual record be copied between the I/O buffer and your program. For read operations, RMS-11 reads a block or bucket into an I/O buffer, finds the desired record within the buffer, and moves the record to a program-specified location.

Before a write or update operation in move mode, your program builds or modifies a record in its own work space. Then, your program issues a write or update record operation request and RMS-11 moves the record to an I/O buffer.

5.2.3.2 **Locate Mode Record Transfers** - RMS-11 supports locate mode record transfer for read operations to all file organizations. However, it permits locate mode on write operations for sequential files only.

Locate mode reduces the amount of data movement, thereby saving processing time. This mode enables your program to access records directly in an I/O buffer. Therefore, there is normally no need for RMS-11 to copy records from the I/O buffer to your program. To allow the program to access a record in the I/O buffer, RMS-11 provides the program with the address and size of the record in the I/O buffer.

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754