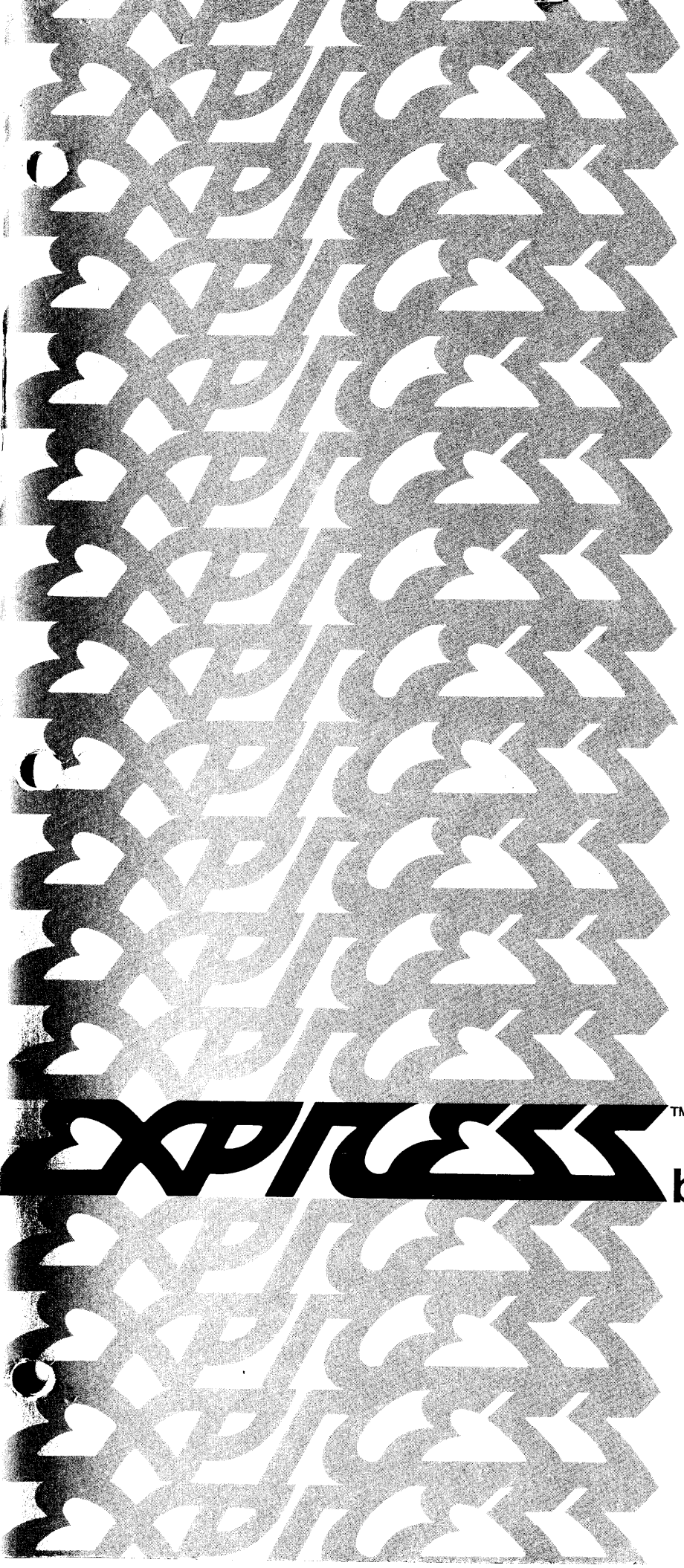


Express™

by Microdata

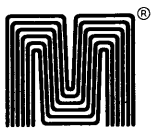
Introduction to Express Multiprogramming Operating System

Price: $5.00

Introduction to Express™ Multiprogramming Operating System

Preliminary
771026A
August 1976

Printed in USA
January 1977
771026A



Microdata Corporation

17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/540-6730 · TWX: 910-595-1764

PREFACE

This manual presents a general overview of the EXPRESS™ Multiprogramming Operating System. It does not describe the specific commands nor does it define system procedure names and calling sequences. For more detailed information, the reader is referred to the following manuals.

1. EXPRESS Multiprogramming Operating System, Shell User's Guide
2. EXPRESS Multiprogramming Operating System, Programmer's Guide
3. EXPRESS Programming Language Reference Manual
4. EXPRESS Programming Language User's Guide
5. EXPRESS BASIC Interpreter Reference Manual

NOTE: This document is a preliminary manual and is subject to major revision.

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

FIGURES

CHAPTER 1

INTRODUCTION

The EXPRESS Multiprogramming Operating System (EMOS) is a general purpose multiprogramming, virtual memory operating system. EMOS is designed to simplify the use of the EXPRESS computer system. It performs the functions of input/output, memory management, file management, process scheduling, and system accounting.

1.1 MAJOR FEATURES

The following paragraphs describe the major features of EMOS.

1.1.1 MULTIPROGRAMMING

A primary feature of the EXPRESS Multiprogramming Operating System is that it allows several processes to be executed simultaneously. This facility of an operating system is called multiprogramming. Multiprogramming is the normal mode of operation of EMOS. There are two fundamental reasons for multiprogramming.

1. Efficiency of system utilization. Most programs run on a computer system do not require all of the resources available. By allowing several such programs to run simultaneously, rather than serially, more efficient use of the system is realized.

2. Multiple access to data. Modern mass storage devices make it possible to keep extremely large amounts of data on-line and available to a computer system (e.g., a small version of the EXPRESS system provides 50,000,000 bytes of on-line storage). Through the use of multiprogramming, many individuals, each at a different terminal, may gain simultaneous access to large data bases.

The number of processes that can be executed simultaneously depends on such factors as the hardware configuration and the resources required by the various concurrent processes.

1.1.2 VIRTUAL MEMORY

The EXPRESS hardware and software provides a virtual memory system. The term virtual memory is used to mean that the size of a program is not limited to the size of main memory actually available in the hardware.

In the EXPRESS system a program is divided into one or more parts called segments. In order for a program to execute, only one of its procedure segments and its data segment needs to be in memory.

The system is designed so that infrequently used segments, such as program initialization routines, are placed in separate segments and normally kept in virtual memory (i.e., they are on the disc). Thus, the amount of main storage required to run programs efficiently is significantly less then the total program size.

The EXPRESS computer system has hardware that recognizes a program reference to an absent segment. When such a reference occurs, an interrupt is generated. Execution of the program is suspended until such time as the virtual memory manager can move the required segment into main memory. In the meantime, the dispatcher allows other programs to continue execution.

1.1.3 RE-ENTRANT CODE

In the EXPRESS system, code segments and data segments are logically and physically separate. The system guarantees that the code segments cannot be altered by normal program execution. Such unalterable segments help mechanize re-entrant code.

The use of re-entrant code means that if a particular code segment is accessible to several concurrent processes, then only a single copy of that segment need be in memory at any one time. If a process using that segment is interrupted, another process may use the same code segment. But since the new process cannot alter the code, when the original process resumes execution, the code segment is still intact and available for use.

The use of re-entrant code also reduces the input/output requirements for virtual memory management. If a segment of code must be overlaid, it is never necessary to move it from main memory to the disc, since the original disc copy is identical to the copy currently in memory.

1.1.4 INPUT/OUTPUT

EMOS treats all input/output operations as file accesses. The programmer accesses a file by an entity referred to as a logical unit. The logical units used in a program are independent of physical devices. The association of logical units with devices is not made until the program is executed. This means that a program may use a different physical file for its input and/or its output each time it is run without changing the program itself.

Programs may also be written that interact with an operator to ask for the name of a file to be operated on.

Files that are physically located on magnetic tape, card readers, terminals, or printers, are accessed sequentially. Files on disc may be accessed sequentially or directly (by specifying a particular record).

Additionally, disc records may be keyed. That is, there may be a key associated with each record of a file. Retrieval from a keyed file may be in the order of the collating sequence of the keys, or specific records may be retrieved by specifying the key.

Records in a file may be fixed length, variable length, or undefined length (stream). Space for disc files is allocated automatically as needed.

1.1.5 SECURITY

The EXPRESS system provides security at two levels. Hardware protection is provided for main memory accesses. This hardware protection prevents one process from accessing a portion of memory allocated to another process.

File security is provided by software and is implemented through security information contained in a file description record associated with each file.

1.1.6 USER INTERFACE

The EXPRESS Multiprogramming Operating System allows the user to interface with the system at several levels of sophistication. At the lowest level, the user need not be aware of the fact that the system is a programmed processor. He need only know the commands necessary to perform the data accessing or data entry functions required by particular applications.

At the next level, the user may interface with the system through the BASIC interface. This interface provides the user with a computational facility. In the simplest case, the user may perform calculations much the same way as he would using a calculator. However, using the BASIC interface, the user may create, edit, and save for future execution, programs of considerable complexity.

A user planning still more complex operations may need to know more about the facilities provided by EMOS. He may have accessing capabilities that allow access to more sophisticated system resources such as task initiation and privileged instruction execution.

1.2 PROCESSING MODES

EMOS provides two basic modes of operation: batch and interactive.

1.2.1 BATCH PROCESSING

The batch processing mode allows the user to submit commands in a single input stream that requests operations such as compilation and execution. Such a stream is called a batch job. Included in the batch input may be source programs and data. Once the user has submitted the batch job, processing of the commands takes place with no further action required by the user.

The input batch may be presented via an input device such as a card reader or magnetic tape unit, or it may be a file previously created and stored on the disc.

1.2.2 INTERACTIVE PROCESSING

In the interactive processing mode the user interacts with the system through the use of a terminal. The interactive mode can be used for program development, data entry, information retrieval, and many other applications where immediate access to the system is required.

The interactive user may create input for a batch job and submit it to be run in batch mode directly from his terminal. He may then continue to perform other interactive operations at the terminal while the system is concurrently executing his batch jobs.

1.3 PROGRAMMING LANGUAGES

The user may prepare programs in any of the following languages:

BASIC
FORTRAN/IV
COBOL
EPL

1.3.1 BASIC

The EXPRESS BASIC language is derived from Dartmouth BASIC. This language is designed to be highly interactive. It allows immediate execution of statements, and includes a simple editing facility. BASIC is not intended for applications that require significant amounts of computation.

1.3.2 FORTRAN/IV

The FORTRAN/IV language may be used for programs that require more computing power than BASIC applications. Because the FORTRAN language was designed before the advent of modern interactive facilities and file systems, programs written in FORTRAN/IV are normally non-interactive and are typically executed in batch mode.

1.3.3 COBOL

COBOL was designed as a business application language. Consequently COBOL provides considerably more file manipulation capability then is provided with either BASIC or FORTRAN/IV. Like FORTRAN/IV, programs written in COBOL are normally executed in batch mode.

1.3.4 EPL

EPL is the primary programming language for EXPRESS. EPL is a modern block-structured programming language providing features such as dynamic storage allocation and recursiveness not available in the other programming languages.

EPL provides programmer access to all of the hardware and software facilities available in the EXPRESS system. Programs may be written in EPL to run either interactively or in batch mode.

1.4 SOFTWARE COMPONENTS

The following paragraphs describe the primary components of the EXPRESS Multiprogramming Operating System.

System Builder	The system builder is a program that runs under control of the operating system. It is used to manufacture a self loading system generator. The system builder may also be used for such functions as system backup and selective file saving.
System Generator	The system generator is a stand-alone program (i.e., it does not run under control of the operating system). The generator is a self-loading module on magnetic tape that creates the disc resident EXPRESS Multiprogramming Operating System.
Memory Manager	The memory manager dynamically allocates main memory and virtual memory among concurrent processes.
Dispatcher	The dispatcher switches the use of the processor among concurrent processes.
Input/Output System	The Input/Output System schedules and performs input and output operations and services device interrupts.
Job and Session Initiation	The Job and Session Initiator allocates a process to a terminal session or batch job.
File System	The file system provides for accessing of data files and maintains file security.
System Librarian	The system librarian maintains frequently used procedure segments that are shareable among users.
Logon Program	The logon program verifies user identification at the beginning of a session.

Shell	Shell is the system command interpreter. It allows the user to run programs. Shell also provides an interactive debugging facility.
Segment Editor	The segment editor accepts language translator generated object modules as input and converts them into executable segments, resolving external references.

1.5 HARDWARE REQUIREMENTS

The minimum hardware configuration required to support EMOS and the programming languages is:

- Central processing unit with 65,536 bytes of main storage.
- 10 million bytes of REFLEX™ disc storage.
- LODESTAR™ magnetic tape cartridge unit.
- PRISM™ video display terminal.
- Line printer.

The following additional hardware may be added to the system:

- Main memory up to 1 million bytes.
- Moving head disc storage.
- Fixed head disc storage.
- LODESTAR tape cartridge units.
- IBM-Compatible reel to reel tape units.
- Line printers.
- Card readers.
- Paper tape readers.
- Paper tape punches.
- Local and remote terminals.

CHAPTER 2

SYSTEM ARCHITECTURE

The EXPRESS computer system was designed with the goals of multiprogramming capability and ease of programming.

The goal of multiprogramming requires a mechanism for keeping only those portions of the active programs in memory that are actually in use. This means that a large program must be divided into smaller units. The two methods normally used for dividing programs into these smaller units are paging and segmenting. With paging, programs are divided into fixed size units along physical boundaries. Thus a single procedure (subroutine) may be split and reside in more than one page. With segmenting, programs are divided into logical units on procedure boundaries. Segments need not be of uniform size.

The EXPRESS system uses the segmentation method. This method has the advantage that the activation and deactivation of a segment can only happen as the result of a transition from one procedure to another from a procedure-call or a procedure-return. Thus, a minimum of hardware is required to recognize these transitions.

Keeping only active segments of programs in main memory requires that swapping of data and programs be made as efficient as possible. For this reason, automatic relocation of procedures and data is provided for in the EXPRESS hardware. When a segment is loaded it may be placed into any available memory location without requiring any changes in the instruction addresses that reference the segment. Not only does automatic relocation mean that segments can be swapped quickly, it also means that a single copy of a procedure can operate on many different sets of data. This ability to share procedures effects a considerable saving in memory requirements.

Another important requirement for multiprogramming is that of minimizing the memory requirements of programs.

Memory space for code is minimized through the design of the instruction set. The instructions of the EXPRESS system are not of uniform length. The most frequently used instructions are one byte (8 bits) long. Data referencing instructions are either two or three bytes long. Instructions occupy consecutive memory locations independent of word boundaries; pad bytes are not required for the purpose of word alignment.

Data space compression is provided for with the EXPRESS system. Storage for data local to a procedure is automatically allocated (i.e., assigned physical memory locations) by hardware when the procedure is activated. The storage is automatically freed on procedure exit. Thus, the same memory may be used by several procedures. Memory for temporary variables, such as intermediate expression results, is also automatically allocated and freed by hardware.

Efficient multiprogramming requires that I/O for one or more processes can occur simultaneously with computations being performed for another process. In the EXPRESS system all I/O devices may move data concurrent with CPU activity.

The EXPRESS system architecture is a departure from the traditional von Neumann type of mini-machine where there is no distinction between program code and data. In the EXPRESS system, code and data are separated into distinct (normally disjoint) segments each with an independent address space (Figure 1). Code segments are non-modifiable; there are no instructions in the hardware for storing into the active code segment. The data segment consists of a static data area and a dynamic data area in the form of a hardware managed push-down stack.

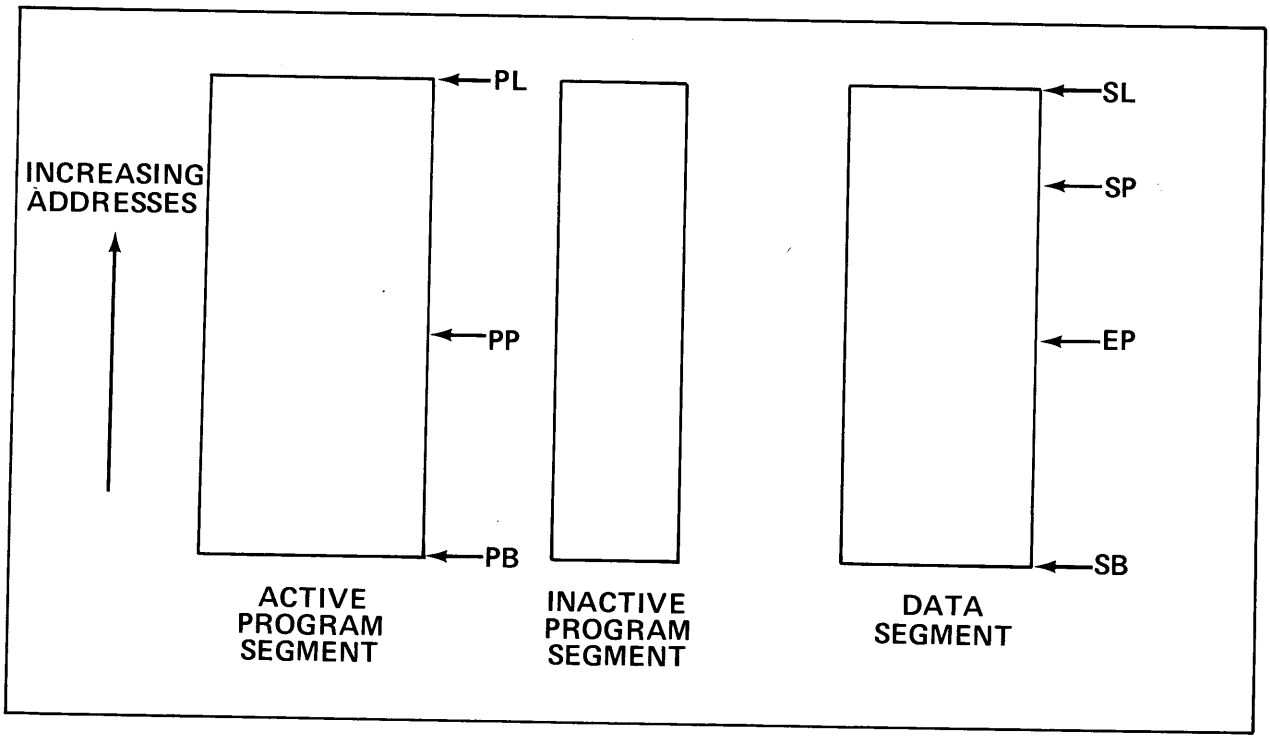


Figure 1. Program Environment

A program consists of one or more program segments and a data segment (Figure 1). A program segment may contain one or more procedures (subroutines). Segments may be of any size up to a maximum of 65,536 bytes. There is a directory called the Program Segment Library (PLIB) that contains an entry for each procedure segment (Figure 2). The PLIB is maintained by the operating system and is used by the hardware for procedure entry and exit. Each entry in the PLIB consists of two words. The first word contains the 16 most significant bits of the 20-bit procedure segment base address (PB). The four least significant bits of the 20 bit address are taken by the hardware as zero. The second word of PLIB contains the 14 most significant bits of the sixteen bit Program Limit (PL). PL defines the extent of the segment. In addition to PB and PL, the PLIB entry contains two bits that are used by the hardware: the Trace bit and the Attention bit. If the Trace bit is set, the hardware generates a call to the TRACE routine after each instruction in the segment has been executed. If the Attention bit is set, an interrupt is generated when a segment is activated as a result of a subroutine entry or exit. The attention interrupt is used to implement the least-recently-used overlay algorithm.

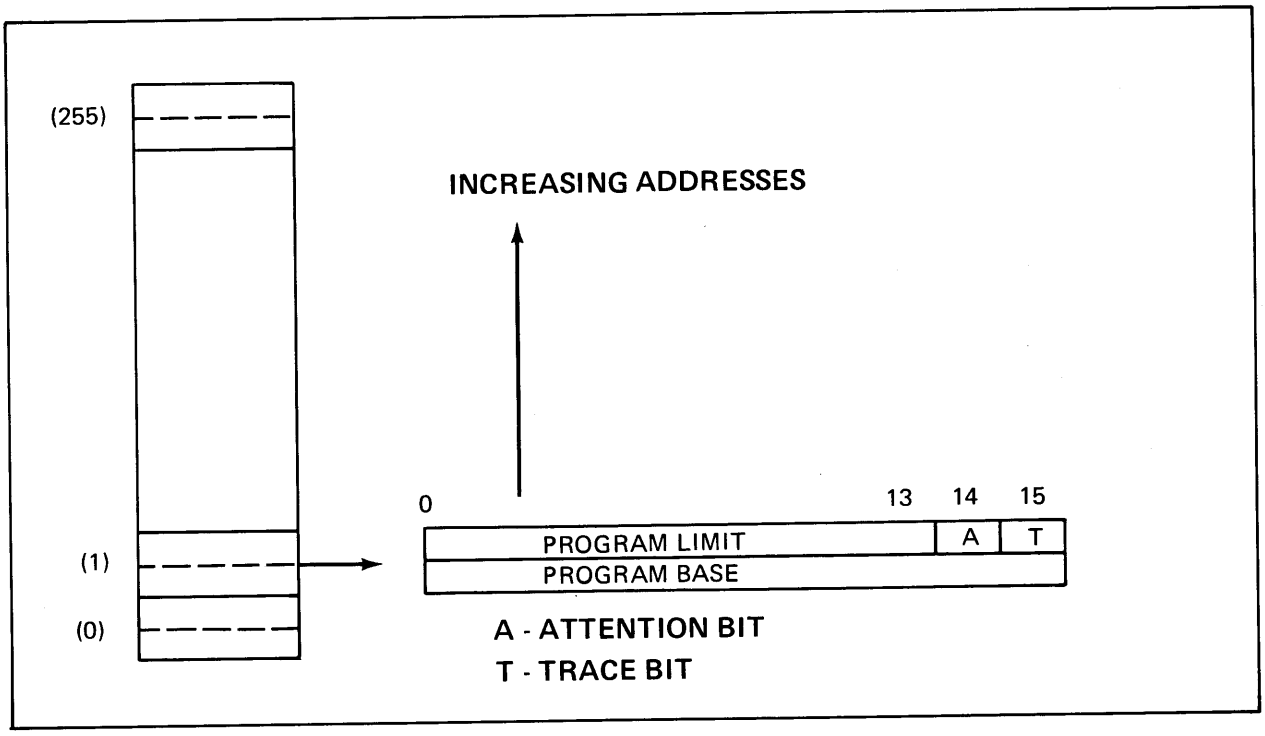


Figure 2. The Program Segment Library

Each Program Segment contains a Procedure Reference Table (PRT) as part of it (Figure 3). The PRT, in turn, contains the entry address of all the procedures whose entries are 'known' to procedures external to the segment. References to external procedures are accomplished by specifying the couple: PLIB-number, PRT-number (PLIBN,PRTN). The PLIBN is used as an index into PLIB to obtain the PB and PL of the referenced segment. The PRTN is then used as an index into the PRT of the segment to determine the actual entry address. The PRT for each segment is created by the segment editor when the segment is bound.

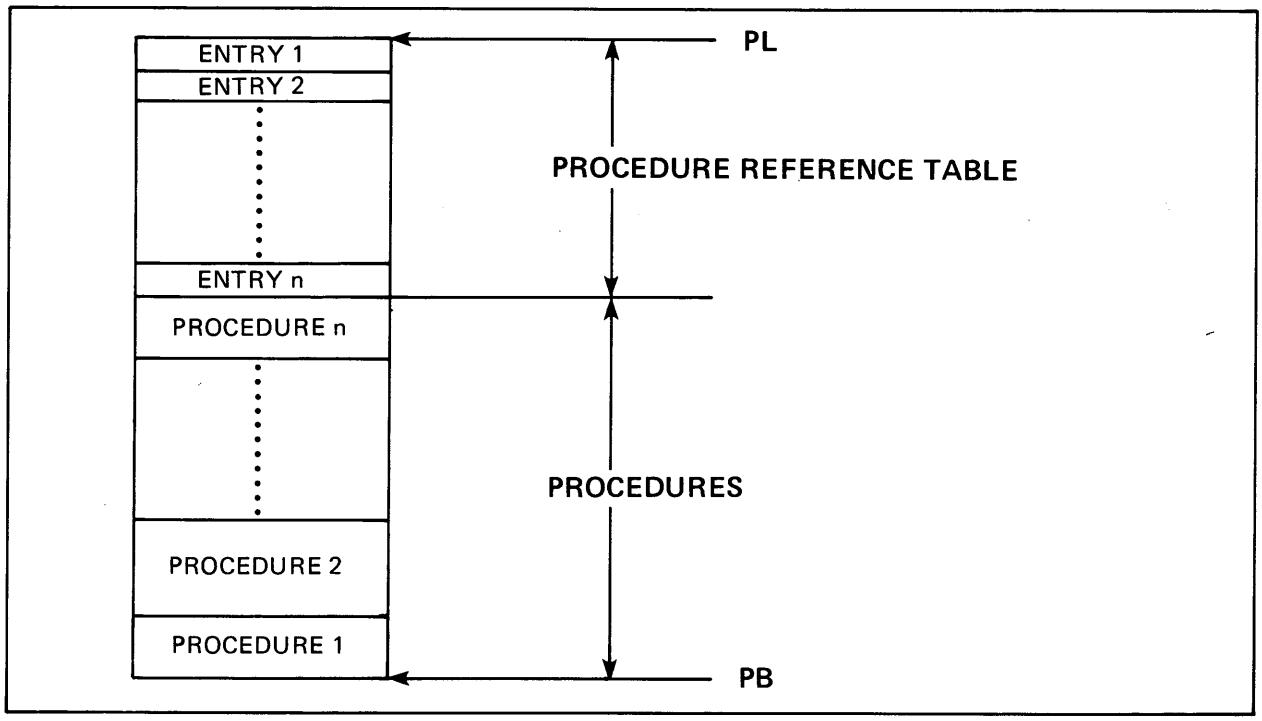


Figure 3. A Program Segment

All addresses appearing in a program are relative. Program branch addresses are relative to the Program Base or to the Program Pointer. The Program Pointer is itself a program base relative address. Data addresses are relative to the Stack Base or to the Environment Pointer. Because the Program Base and the Stack Base are distinct, branch instructions and data referencing instructions containing the same address will actually reference different physical memory locations.

The data segment is organized into the form of a push-down stack that is automatically maintained by the hardware (Figure 4). The stack provides for dynamic storage allocation, automatic recursiveness, an automatic and systematic mechanism for subroutine linkage and argument passing, automatic saving of the environment at the occurrence of an interrupt, and a simple and efficient mechanism for switching the central processor from one process to another.

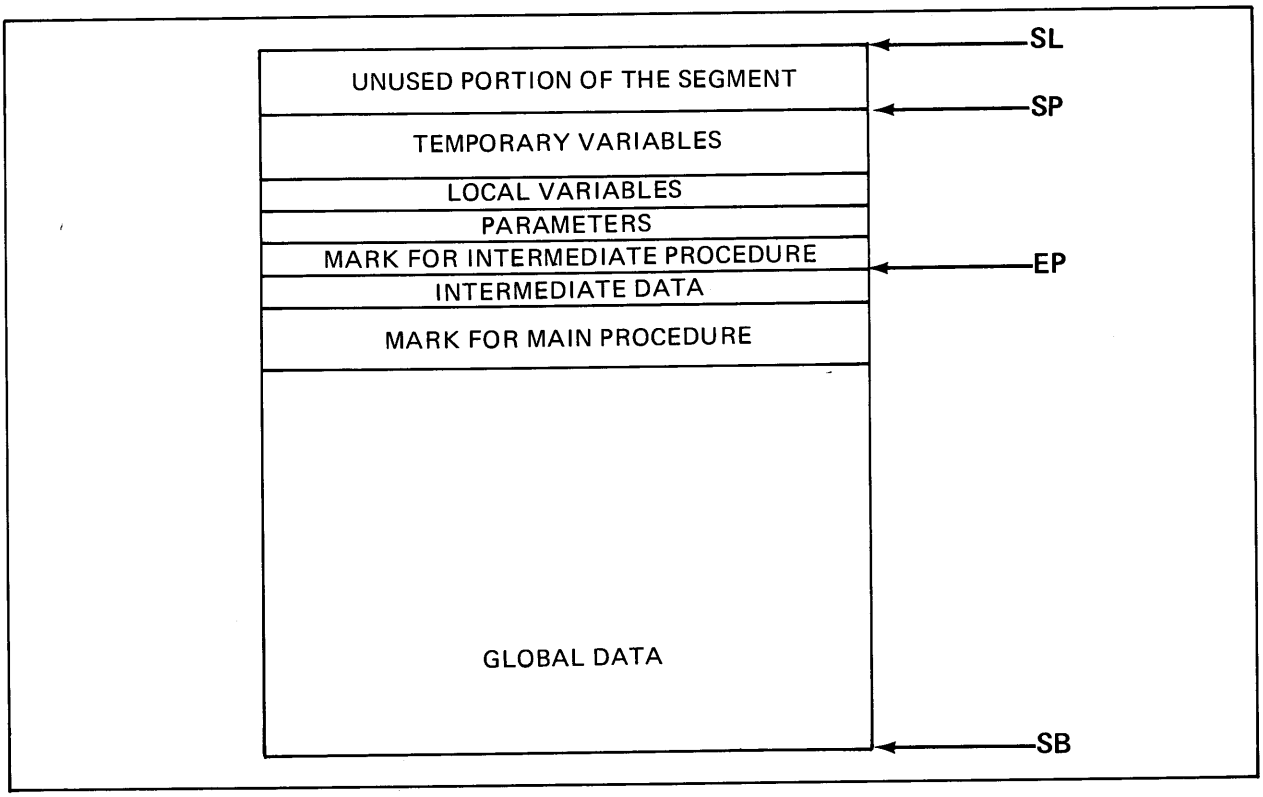


Figure 4. Stack Organization

Each time a procedure is called, a mark is placed into the stack. The mark contains the information for changing the environment back to the calling program upon procedure exit. Marks are also placed into the stack automatically as the result of an interrupt.

The Express System has two types of interrupts; internal and external. Internal interrupts occur as a result of conditions in the currently executing process; for example, an attempt to execute a privileged instruction or an attempt to access data outside the current data segment. External interrupts occur as the result of an event outside the currently executing process; for example, the completion of an I/O operation scheduled by some other process.

When an external interrupt occurs, the hardware automatically places a mark in the stack of the current process, thus saving its environment. Next, the interrupt procedure is invoked. At the end of the interrupt procedure, the hardware automatically turns control over to the Dispatcher. This portion of the operating system then resumes execution of the highest priority process that is now ready. This may or may not be the process that was interrupted.

Programming ease on the Express system is achieved through the use of EPL. EPL is an easy to use language that makes effective use of the hardware facilities provided by the Express system. EPL replaces the low level assemblers normally used on other machines. All system software for EXPRESS is written in EPL (this includes the compiler for EPL).

CHAPTER 3

ACCESSING EMOS

3.1 USER CAPABILITIES

Information concerning each user of EMOS is stored internally in the User Capability File. The System Manager controls this file and may add and delete users. He also assigns the capabilities to the user.

The user capabilities specify the particular system services available to the user, and the files that are accessible to the user, along with the type of file access allowed. For example, a user may be allowed to access a file containing a program but only for the purpose of execution, while some other user may be allowed to modify the same program file.

3.2 LOGGING ON TO THE SYSTEM

When a terminal is activated, the system creates a process that initiates the LOGON program. The LOGON program requests the individual at the terminal to identify himself. The identity is checked by looking through the User Capability File. If the individual is an authorized user, the LOGON process determines the program that the particular user is to run. This program is normally SHELL (a general-purpose user command interpreter) or BASIC (the BASIC language interpreter). However, the User Capability File may specify that the individual is to run a particular application such as a data entry program.

3.2.1 THE SESSION

Once a user has successfully logged on to the system, his subsequent interaction with the system is called a session. The session exists until the user logs off of the system. A session may also be inadvertently terminated. For example, if the terminal is connected through a communication link, an interruption in the link will terminate the session. During a session the user may execute a number of operations allowed by the user's capability list. As an example, a session with a user having normal programming capability might consist of the following operations:

1. Run the text editor to create or modify a source program.

2. Submit the program created by Step 1 to the background batch process for compilation and execution.

3. Run the text editor to create another source program.

4. Run the compiler as a session task to compile the program created in Step 3.

5. Run the compiled program using the on-line debugging facility.

6. Save the compiled version of the program for subsequent execution.

7. Log off.

Note that the program submitted in Step 2 above may or may not have been completed before the user logged off the system.

3.3 THE SHELL

SHELL is the command interpreter that is normally invoked by the LOGON programs for users who are doing program development. SHELL communicates with the person at the terminal by accepting commands. As a result of the commands, SHELL may cause a series of subtasks to be executed; e.g., compile, do segment linking, and run the linked program. In setting up a subtask, SHELL may establish a connection between the subtask and one or more files. Such connections can be established as a result of explicit SHELL commands or as a result of initialization information stored with the program to be run. This is accomplished by connecting the files to SHELL and allowing the subtask to share the file. Passing of files from one step to the next is accomplished by connecting the file to SHELL and allowing shared access by each subtask in turn.

3.3.1 THE SHELL COMMANDS

SHELL normally accepts commands from the terminal, one at a time, checks the command for validity, and causes the proper action. Additionally, the user may create a file containing a list of command images and instruct SHELL to accept commands from that file.

The type of action taken by SHELL when it receives an erroneous command depends on whether the command was input from the terminal or a command file. If the command was input from the terminal, an error message is displayed and the user is prompted for another command. If the command was read from a file, the command along with an error message is displayed on the message file and the commands in the command file are ignored.

The following paragraphs summarize the types of commands that are available under SHELL.

3.3.1.1 Program Execution Commands

The program execution commands allow the user to execute previously created programs. These programs include the system programs such as the Text Editor, the EPL, FORTRAN, and COBOL compilers, etc., as well as user created programs.

With the use of the program execution commands, the SHELL user can invoke the BASIC Interpreter Subsystem.

3.3.1.2 File Maintenance Commands

This group of commands is used to save and delete files. The files may be object modules generated as the result of a compilation, or they may be arbitrary text files. These commands also control file accessibility rights.

3.3.1.3 File Connection Commands

These commands are used to connect or disconnect a file to a program. See Chapter 4 - THE FILE SYSTEM for a description of the concept of a connection.

3.3.1.4 Debug Commands

The debug commands provide interactive debugging capability. The facilities provided with these commands are:

1. Interpretive Execution of Portions of a Program. The facility makes use of the hardware trace feature. Using this facility allows programs to be executed one instruction at a time displaying the results of each instruction. Additionally, a program may be controlled to execute until one of a list of breakpoints is reached with or without instruction results being displayed.

2. Dumping of Data and Procedures. Any portion of the user's data segment and/or procedure segments may be dumped.

3. Mark Display. The marks in the stack may be displayed. This is done to show the dynamic procedure calling history.

4. Expression Evaluation. Expressions can be evaluated. Constants in expressions can be input as decimal or hexadecimal values, and the result can be displayed in either decimal or hexadecimal.

3.3.1.5 Utility Commands

The utility commands provide the following functions:

1. Display current time and date.

2. Show the status of the session or of the background batch process.

3.3.1.6 Interrupt Commands

When SHELL is not in command accept mode, certain control characters can be used to interrupt the current subtask. The interrupts performs the following functions:

1. Enter Debug Mode. This interrupt allows the user to use the debug commands to inspect the status of a program. He may then resume execution, resume in interpretive execution mode, or he may abort execution of the subtask.

2. Interrupt Listing. This interrupt is recognized by certain System Programs such as the Text Editor. This interrupt causes the listing operation currently being performed to be aborted. However, the execution of the program continues.

3.4 USER CODED COMMAND INTERPRETER

EMOS provides the user with the ability to write his own command interpreter. In fact, SHELL is a user level program. The difference between SHELL and other user programs is that SHELL is invoked by LOGON. If the user wants to provide a different command interpreter, say, X, he may do so by writing the Program X. This program is then entered into any (or all) accounts of the user capability file as the program to be activated by LOGON.

CHAPTER 4

THE FILE SYSTEM

One of the major components of EMOS is the set of intrinsic procedures referred to collectively as the file system. The file system allows the user's program to perform I/O at a logical rather than a physical level. This means that the user need not concern himself with the physical characteristics of I/O devices.

4.1 CONNECTIONS

Within a program, files are referenced indirectly by using a file name. This file name is actually the name of the control block that is used by the file system intrinsic procedures to coordinate and control the I/O activity. The control block contains information about the physical device and about the buffers associated with the device.

Prior to performing I/O activity on a file, there must be an association made between the file name and an actual data set. This association is called a connection.

The connection can be made either before the program begins execution through the use of SHELL commands, or can be made by the program itself after execution begins. Each time a program is run, a given file name can be connected to a different physical data set. As an example, an inventory update program may refer to the file that it processes by the file name 'UPDATEFILE'. Then each time the program is run it may request the operator to name an inventory data set to be updated. On the first run the user may specify 'PARTSINVENTORY' as the data set name and on the second run he might specify 'EQUIPMENTINVENTORY'. In each case the program calls upon the CONNECT system intrinsic passing the actual data set as a parameter. Then the program uses the name 'UPDATEFILE' to perform all file I/O operations.

4.2 CATALOGED FILES

The names of all permanent disc files are maintained in the System Catalog. The Catalog is an hierarchically structured directory. A directory is a special file whose keys are a data set and whose records are data set descriptions.

The description (location on the disc, etc.) of the root of the directory is known a priori to the file system connection intrinsic. To locate a file cataloged in another directory, a path name naming in order the directory nodes required to reach the required file, must be given.

When a user logs on to the system, the path name for the directory of his 'private' files is made known to SHELL. Thus, to reference a user's private files, only the last name in the path is required in order to form a connection to the file.

The user need not be aware of the internal organization of the catalog. System services are provided to catalog files and delete files.

4.3 FILE ACCESS AND SECURITY

In order to access a cataloged file, the user must:

1. Know the name of the file.

2. Know the path name to the directory in which the file is cataloged. In the case of the user's private files, the path name to his directory is known to SHELL and need not be known to the user.

3. Have the authority to perform the type of access requested.

File access security information is stored in the file descriptor and is checked by the connection intrinsic. If the requesting program does not have the authority to access the file, the connection will not be made.

The account under which a file is created is termed the owner account. Only the owner account and the System Manager account are permitted to alter the access information contained in the file description.

The owner can control access privileges for three classes of user accounts.

1. The owner's account.

2. A list of accounts, specified by the owner.

3. All other accounts.

The owner may specify the same or differing access privileges for each class. For most files, the owner account (Class 1) will have total accessing privileges and the other classes will be denied any access to the file.

The permitted accesses are based on the fundamental actions of:

Read	Read data from a file
Append	Add new records to a file at the end
Write	Change the contents of a file, including record deletion
Execute	Execute a program contained in a file
Probe	Search through a directory
Alter	Modify a directory

4.4 DISC FILE ORGANIZATION

Each file on the disc consists of two components, data records and record indexes. Each index points to a single data record and there is exactly one index per record. There are two distinct index types: position and key (see Figure 5). The two index types result in two distinct file structures; positional and keyed. A positional file record consists only of data, a keyed file consists of a key and data; both have indexes to speed access.

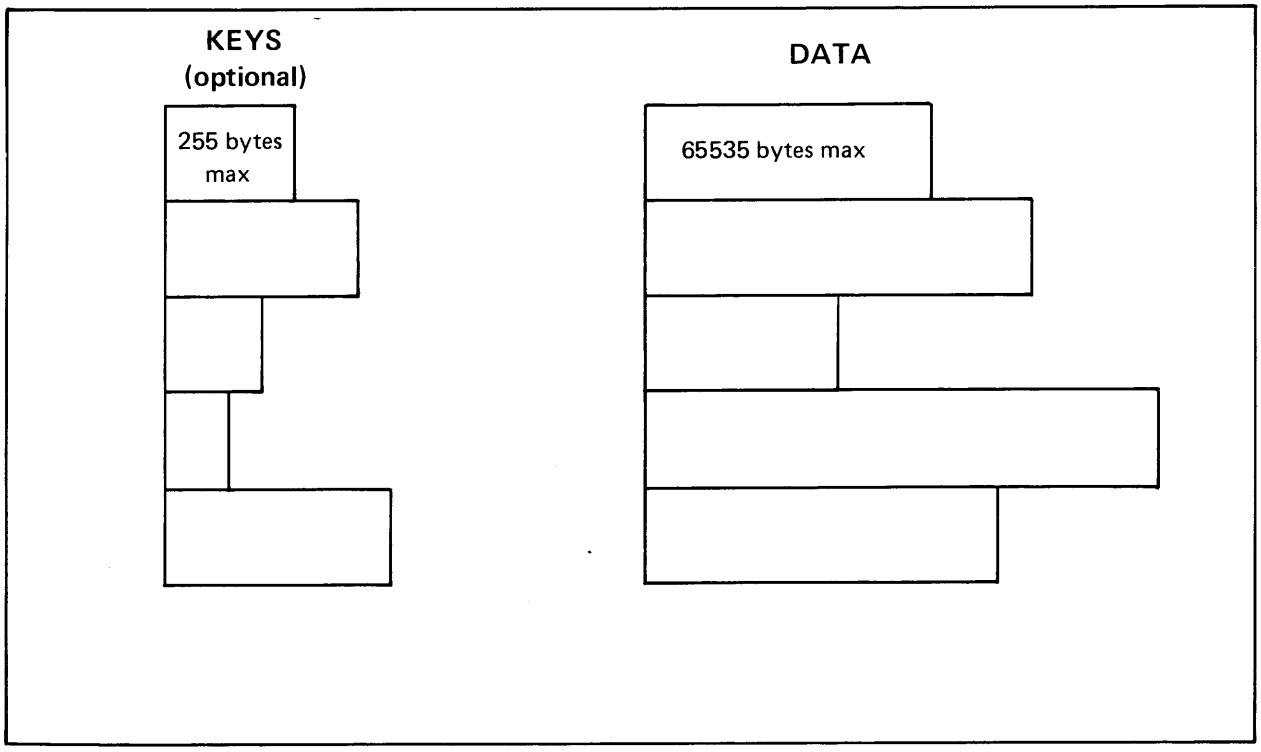


Figure 5. Disc File Structure

Each file structure has an access method for sequential transaction ordering and for random transaction ordering. Either file structure allows records of arbitrary length. Records may be inserted, deleted, expanded, or contracted at any point in the file. The current record position may be altered forward or backward at any time. In particular, forward space, backspace, position-to-start, and position-to-end are allowed.

Indexes are part of every file. They are visible only to the file system intrinsics and are removed from the user's concern. Also record blocking and file space allocation are removed from the user's concern. Files grow and shrink as records are added and deleted. The user never need concern himself with space allocation for his files.

4.5 ACCESS METHODS

After a connection is established, access to data may be by any of four methods; serial, direct, collated, or content. For connections to data sets on devices other than disc, only serial access is permitted. (See Figure 6).

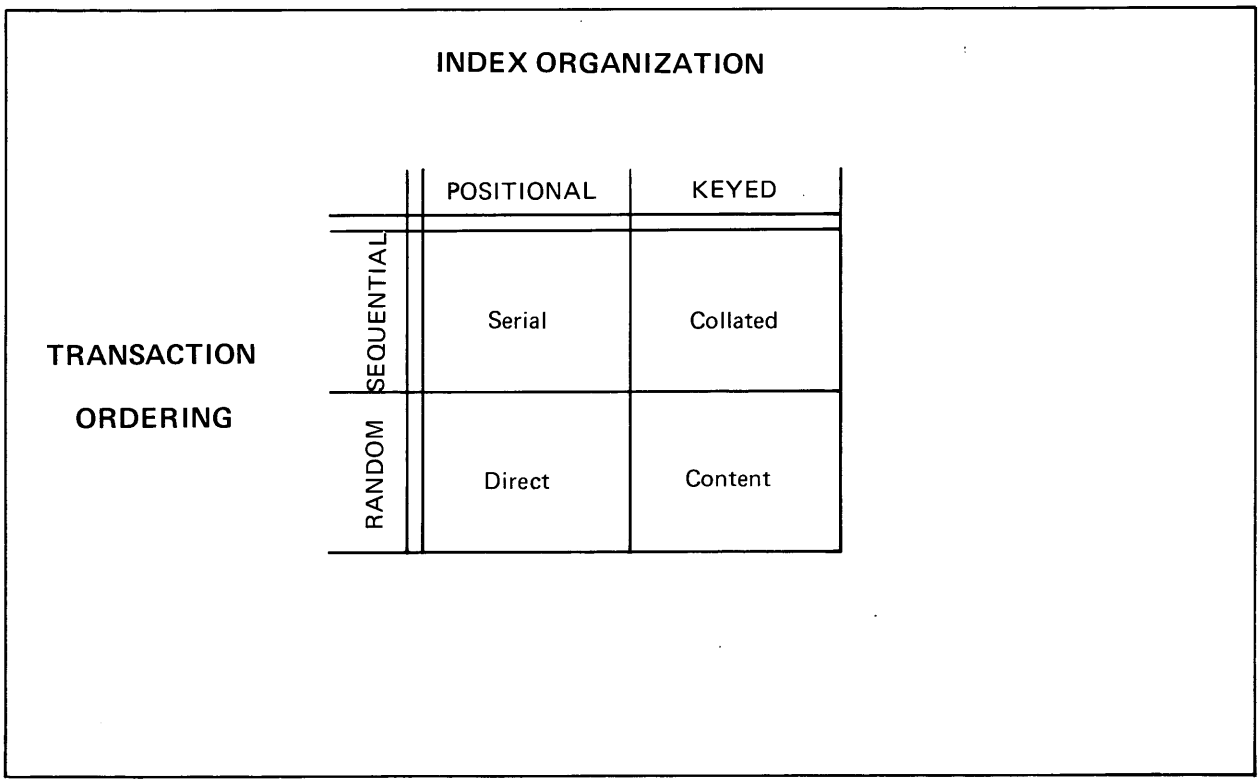


Figure 6. Access Methods

4.5.1 SERIAL ACCESS

Serial accessing applies to positional files on disc and to files on sequential devices such as card readers, printers and magnetic tape.

When positional file records are written, an index to the record is created. When records are retrieved from a positional file using serial accessing, they are retrieved in positional order.

4.5.2 DIRECT ACCESS

Any record in a positional file may be directly accessed by specifying a logical record number. The logical record number is implied by the position of the index entry. Insertions and deletions are allowed for positional files. These operations alter the logical record number of records following the position of the insertion or deletion.

4.5.3 COLLATED ACCESS

Records in a keyed file may be thought of as being in sequence by key name. With collated access the next record retrieved is the one whose key is next in the collating sequence. The collating sequence used to define key order is the ASCII sequence.

4.5.4 CONTENT ACCESS

Records in a keyed file may be accessed by key name. If there is no record with the given key name, the file is positioned to the first record whose key is higher in the collating sequence and no record is retrieved.

4.6 FILE SYSTEM COMPARISON

The Express File System offers accessing capability comparable to that of large data management systems. Because the IBM OS and IBM VS system are well known, a comparison with these systems is made here.

IBM OS data sets have four possible organizations:

1. Sequential
2. Indexed sequential
3. Direct
4. Partitioned

IBM VS supports two additional data set organizations:

1. Keyed sequenced

2. Entry sequenced

Figure 7 compares IBM organization and access method with Express equivalents.

IBM Organization	IBM Access Method	Express
Sequential	BSAM, QSAM	Serial access on a positional file
Indexed sequential	BISAM, QISAM	Collated or content access on keyed file.
Direct	BDAM	Direct access on a positional file
Partitioned	BPAM	A directory of positional files
Entry Sequenced	VSAM	Positional
Key Sequenced	VISAM	Keyed

Figure 7. Comparison of Access Methods

IBM access method names are preceeded by letters to indicate the access technique. B for basic, Q for queued, and V for virtual. Basic accessing involves no system record blocking and no I/O event synchronization, meaning that the user must provide these operations in his program. Queued accessing provides record blocking and I/O synchronization by the system. Virtual access allocates and moves data in units of virtual memory pages.

Express I/O to non-disc files is serial and supports all features of IBM sequential I/O to such files. Express disc file I/O supports most IBM features and is less cumbersome. Buffering, space allocation, blocking, addressing, and reorganization are handled by the system rather then requiring user involvement.

IBM disc space must be allocated by the user in contiguous blocks before using a file. The Express File System allocates space dynamically as it is needed. IBM Job control language requires the user to be concerned with physical block sizes. Express allows a record to span many blocks and performs all blocking of logical records.

Express positional index files combine features of IBM sequential and direct data sets. Records may be inserted, deleted or randomly selected on the bases of record number. Records may be read in reverse order which is not possible with IBM methods. The Express user works with logical record numbers. The file system handles actual disc addresses. IBM methods require the user to deal with disc addresses, making insertions difficult.

CHAPTER 5

USER PROGRAMS

This chapter describes the mechanisms for creating and executing user written programs. The discussion applies to EPL, FORTRAN, and COBOL programs. The maintenance of BASIC programs is accomplished directly by the BASIC interpreter. For details see the "Express BASIC Interpreter Reference Manual".

5.1 CREATING A LOAD MODULE

Several steps are required to prepare a program for execution. The first step is the compilation of the source program. The compiler generates output called object modules. These object modules are specially formatted files that are used as input to the Segment Editor. Object files are normally cataloged in the user's directory.

The next step takes each of the object modules that comprise the user's program and converts it into a load module. A load module consists of one or more code segments and an encoded set of instructions for creating a data stack. The conversion from object module to load module is performed by the Segment Editor. The primary function of the Segment Editor is to resolve all external data and procedure references and to create a procedure reference table for each segment. The load module, created by the segment editor, is also normally cataloged in the user's directory.

Once the load module has been prepared, it may be executed at any time by using the SHELL run-command. At this time, appropriate connection commands may be given to override possible default connections defined in the load module. When the run-command is given, the load segments are moved from the user's file into virtual memory. At this time, references to system library procedures are resolved and the library procedures are also moved into virtual memory, if they are not already there. The result is a program that is scheduled for execution based on its priority.

5.2 THE SEGMENT EDITOR

The Segment Editor's primary function is to combine compiler generated object modules into a program module. This is accomplished in the following way. The user specifies the name of the main program. He may also specify any number of directories to be searched. If no directories are specified, only the user's directory and the system public directory are used. The Segment Editor reads the object module converting it into program module form. If this module references any other object modules, these are searched for in the directories specified by the user, and in the order specified. This process continues until all the referenced object modules (except System Intrinsics) have been bound into the program module. System Intrinsics are not bound into a user segment. Instead, they are bound into public, shareable segments.

Commands are provided to the Segment Editor by the user so that the user may segment the program so as to improve its run time efficiency.

CHAPTER 6

THE SYSTEM KERNEL

The Kernel is the minimum part of EMOS required to run programs. The components of the Kernel are:

1. A dispatcher that switches the use of the processor among processes.

2. A virtual memory swapper that switches the use of main memory among processes.

3. The device initiate and interrupt routines.

4. The time-of-day clock manager that organizes and signals time events.

5. A job/session initiator that allocates a process to a terminal session or to a background batch job.

6. A set of 'operations' that accomplish process synchronization, process communication, and resource management.

7. A set of 'requests' that provide a protected interface between slave mode programs and master mode services.

8. A set of intrinsic routines that support the file system.

6.1 PROCESSES

A process can be thought of as a program in execution. A process is said to be active if it has not yet terminated its execution. Many processes may be active simultaneously, although at any instant only one process can be running. The running process is the one whose instructions are being executed by the CPU. The design of EMOS is process oriented and the Kernel deals exclusively with processes.

A process consists of a set of private data spaces, the stacks, and active process descriptor blocks. When the process is not running, information about which code segment and which instruction in that segment will next be executed, is contained in marks in the stacks for the process. When the process is running, this information is contained in the hardware registers of the CPU. Note that code segments are not owned by a process, but are merely used by them. This is the reason that code may be shared among processes.

When a terminal is activated, a process is created whose function is to communicate with the terminal. This process initially runs the LOGON program which then proceeds to communicate with the terminal user.

When users write programs they need not be aware of the process structure. Their programs cannot exercise control over it. A user may write a program being completely unaware of the multiprogramming aspects of the system. Subsequently, this program can be shared by many users simultaneously.

Within EMOS, processes are logically organized in a tree structure. This means that a process may have only one immediate father (the creator of the process) but it may have any number of sons (processes created by the process).

There exists one process which is the root of the process tree. This process is called the grandfather process.

When EMOS is loaded into the Express Computer by the IPL button, the only process running is the grandfather. This process immediately creates a set of system service processes, a process for each I/O device, and finally a process that monitors terminals and batch input devices. This latter process notifies the grandfather when a terminal or batch input device is activated.

6.1.1 TASKS

The grandfather process creates two types of processes: system service processes, and user service processes. System service processes do not create son processes. User service processes may create one or more son processes. These son processes are referred to as tasks. A task may be considered to be a co-routine of its father in the sense that when the son task is making progress, its father cannot make progress and while the father tasks are making progress, the son cannot make progress.

6.2 PROCESS STATES

Processes once created exist in one of several states:

1. The Running State. The process whose instructions are currently being executed by the CPU. At any instant in time only one process can be running.

2. The Ready State. A process is ready if it could run given the CPU.

3. The Wait State. A process places itself in the wait state if it requests a resource not immediately available (e.g., a record from an I/O device).

4. The Suspended State. A process may be suspended from execution by its father. Suspended processes can only be reactivated by their father.

6.3 PROCESS SCHEDULING

A primary function of the Kernel is to dispatch ready processes. This is accomplished by placing the ready process descriptor block in a particular ready queue based on the priority of the process. The dispatcher then runs the process at the head of the highest priority queue. The descriptor block contains a pointer to the particular task to be run.

Once a process is running, it will continue to run until one of four events occurs:

1. The process completes execution.

2. The process places itself in the wait state by requesting an unavailable resource (normally an I/O request).

3. The time quantum allocated to processes on this queue has been used. The process then moves to the front of the queue. Some queues have infinite time quantums. Processes assigned to such queues are completed in FIFO order.

4. A process in a higher priority queue than that of the currently running process, becomes ready. This can happen as the result of an external interrupt.

6.4 KERNEL LEVELS

There are two levels of abstraction in the Kernel. Each level is identified by its list of primitives. Level 1 primitives are called 'operations'. Level 2 primitives are called 'requests'. The user is normally not concerned with these levels of the Kernel. User programs normally interface with the system at a higher level; i.e., the File System intrinsics.

6.4.1 LEVEL 1

The objects that exist at this level are processes semaphores (process synchronization queues), interprocess message queues, and I/O buffers. Level 1 'operations' at this level run in master mode since they must do absolute addressing (reference data outside a stack) and execute other master mode instructions.

6.4.2 LEVEL 2

The objects that exist at this level are tasks, connections, programs, and files. The 'requests' manipulate these objects in order to implement the File System intrinsics and provide other user program services.

Microdata Corporation
17481 Red Hill Avenue, Irvine, California 92714
Post Office Box 19501, Irvine, California 92713
Telephone: 714/540-6730 · TWX: 910-595-1764

PRINTED IN USA 771026A