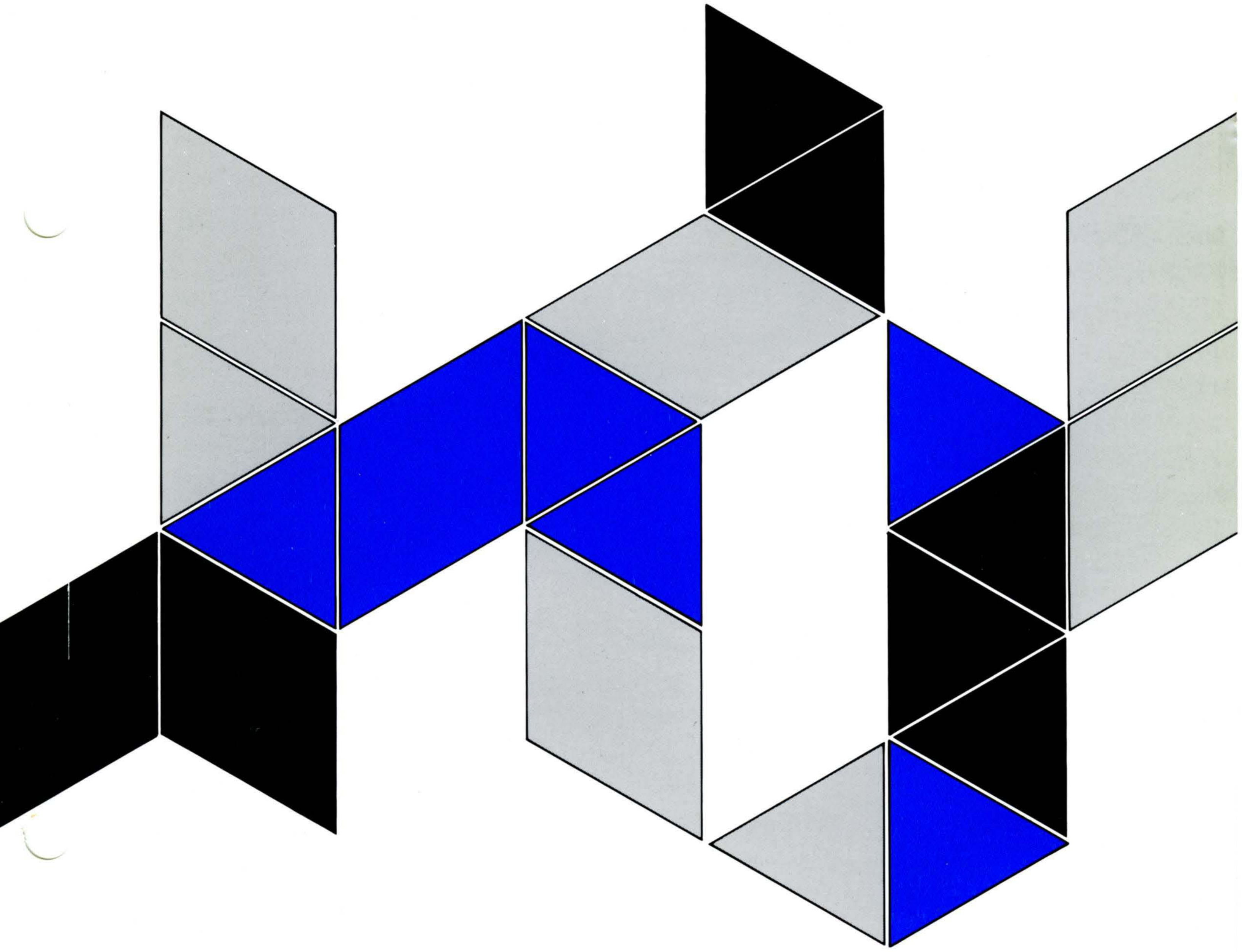
IBM

Series/1

Realtime Programming System Version 6: Concepts and Facilities

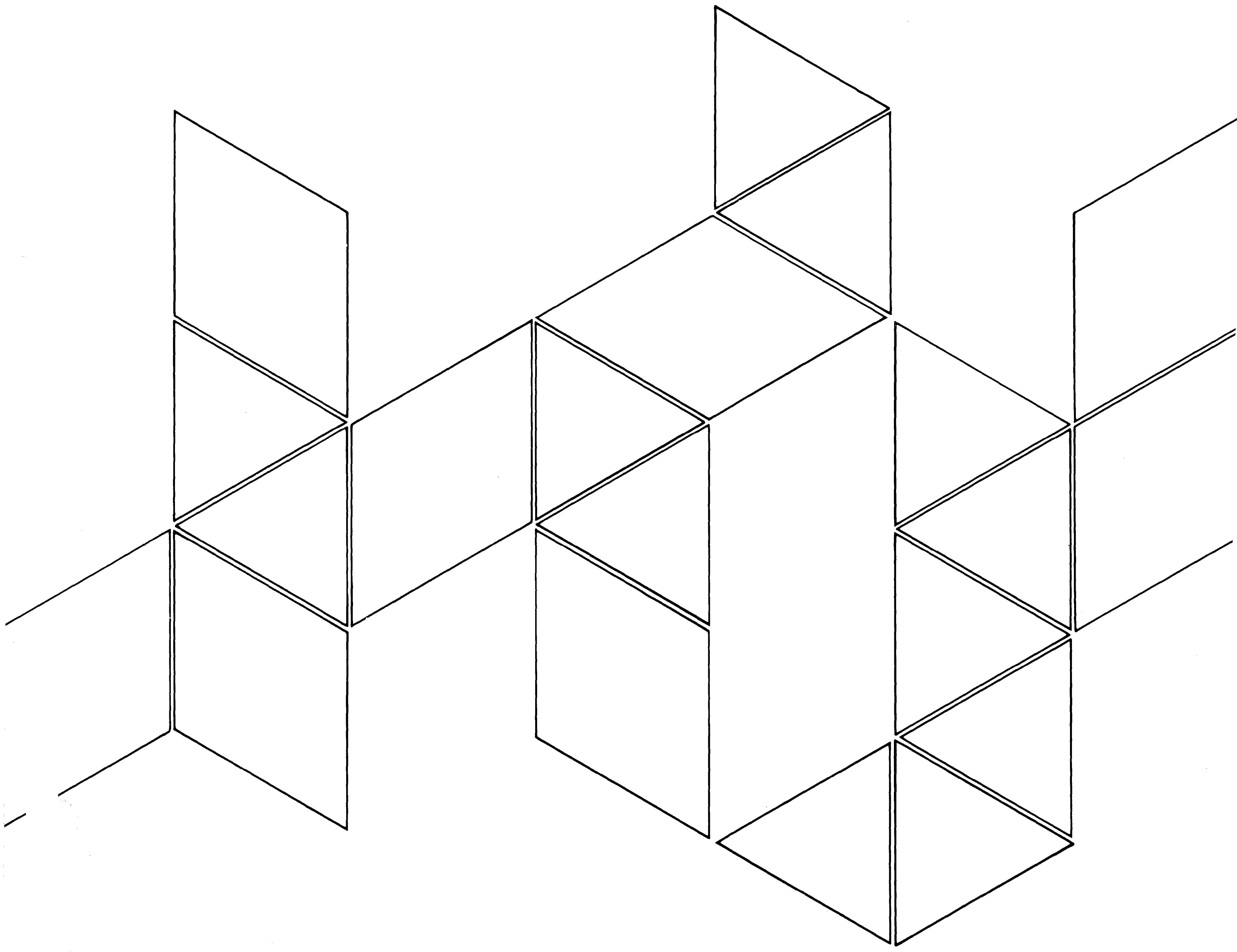


IBM

Series/1

Realtime Programming System Version 6: Concepts and Facilities

GC34-0471-1
Program Number 5719-PC6



Second Edition (February 1984)

This is a major revision of, and makes obsolete, GC34-0471-0 and Technical Newsletter GN34-0840.

This edition applies to Version 6 Modification Level 2 of the IBM Series/1 Realtime Programming System (Program number 5719-PC6). Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Preface

This book describes Version 6 of the Realtime Programming System — a general purpose operating system for the Series/1. The purpose of this book is to:

- Present system facilities and indicate how the Realtime Programming System implements general operating system concepts
- Discuss licensed programs that extend the functions of the base operating system
- Provide introductory information to help you understand how the operating system works

How This Book Is Organized

This book contains five chapters.

- Chapter 1, "Overview of the Realtime Programming System" highlights the Realtime Programming System and briefly describes the types of applications and the licensed programs that the Realtime Programming System supports.
- Chapter 2, "Operating System Facilities" describes the structural components (facilities) of the operating system. Such topics as multiprogramming, multitasking, system I/O, and file management are discussed.
- Chapter 3, "Data Communications and Network Support" discusses the Realtime Programming System support for data communications devices, SNA networks, and Series/1 networks.
- Chapter 4, "User Interfaces to the System" tells users how to use the operating system.
- Chapter 5, "Development Tools and Application Aids" describes high-level programming languages and the tools IBM provides for writing programs on a Series/1.

Related Publications

Related publications are publications that discuss or relate to the subjects described in this book. The following list states the complete titles of this manual's related publications. For the sake of brevity, the related publication titles referred to later in this book do not include the company, system, version designation, or order number.

The library for the Realtime Programming System consists of:

- The Realtime Programming System manuals
- The Program Preparation Subsystem manuals
- Manuals for other licensed programs (including programming RPQs) that run under the Realtime Programming System
- Miscellaneous Series/1 manuals

Related publications are as follows:

- *IBM Series/1 Realtime Programming System Version 6*:
  - *Binary Synchronous and Start-Stop Communications Programming Guide*, SC34-0478
  - *Command Language Facility User's Guide*, SC34-0462
  - *Configuration Guide*, SC34-0562

- *Control Blocks*, SC34-0472
- *Data Management Programming Guide*, SC34-0468
- *Glossary and Subject Index*, GC34-0473
- *High-Level Language System Services Reference*, SC34-0537
- *Macro Reference*, SC34-0463
- *Messages and Codes*, SC34-0464
- *Network Definition Utility User's Guide*, SC34-0546
- *Operation Guide and Reference*, SC34-0565
- *Problem Determination*, SC34-0470
- *Reference Summary*, SX34-0110
- *Standard System Installation Guide*, SC34-0467
- *Supervisor Services Programming Guide*, SC34-0469
- *System Customization Guide*, SC34-0466
- *System Planning Guide*, GC34-0489
- *Systems Network Architecture Support Installation Guide*, SC34-0512
- *Systems Network Architecture Support Programming Guide*, SC34-0511
- *Utilities Reference*, SC34-0465

• *IBM Series/1 Program Preparation Subsystem Version 6*:
- *Application Builder User's Guide*, SC34-0475
- *Batch User's Guide*, SC34-0476
- *Macro Assembler User's Guide*, SC34-0477
- *Text Editor User's Guide*, SC34-0474

• *IBM Series/1*:
- *-System/370 Channel Attach Program General Information Manual*, GC34-0217
- *-System/370 Channel Attach Program Reference Manual*, SC34-0215
- *Advanced Remote Job Entry User's Guide*, SC34-0452
- *COBOL Language Reference Version 2*, GC34-0392
- *COBOL Programmer's Guide Version 2*, SC34-0394
- *Communications Manager for the Series/1 Introduction*, GL23-0060
- *Fortran IV Language Reference*, GC34-0133
- *Fortran IV User's Guide*, SC34-0134
- *Indexed Access Method Version 2: User's Guide*, SC34-0396
- *Job Stream Processor Programming RPQ P82625 User's Guide*, SC34-1716
- *Mathematical and Functional Subroutine Library User's Guide*, SC34-0486
- *Multiple Terminal Manager Version 3 General Information Manual*, GC34-0509
- *Multiple Terminal Manager Version 3 User's Guide*, SC34-0455
- *Pascal Programming RPQ, P82658, P82659: Language Reference*, SC34-1731
- *Pascal Programming RPQ, P82659: User's Guide*, SC34-1729
- *PL/I Language Reference*, GC34-0085
- *PL/I User's Guide*, SC34-0086
- *Query General Information Manual*, GC34-0457
- *Query Installation and Programmer's Guide*, SC34-0427
- *Query User's Guide and Work Book*, SC34-0428
- *Remote Manager User's Guide*, SL23-0097
- *Sort/Merge Programmer's Guide*, SL23-0011
- *Systems Network Architecture Remote Management Utility Programming RPQ P82639 User's Guide*, SC34-1712
- *X.25/HDLC Communications Support: Programming and Operating Reference Manual*, SC09-1029
- *4987 Programmable Communications Subsystem Extended Execution Support Reference*, SC34-0187
- *4987 Programmable Communications Subsystem Preparation Facility Reference*, SC34-0119

- *IBM Series/1*:
  - *Digest*, G360-0061
  - *Principles of Operations*, GA34-0152
  - *Software Service Guide*, GC34-0099
  - *System Selection Guide*, GA34-0143

Problems with the Program

If you have a problem with a Series/1 program, see the *IBM Series/1 Software Service Guide*, GC34-0099, for instructions on how to report the problem and obtain resolution.

Contents

Figures

Summary of Amendments

Summary of Amendments
for *Concepts and Facilities*, GC34-0471-1
Realtime Programming System Version 6 Modification Level 2

Additions to this Book

- Support for the Programmable Two-Channel Switch enables a set of common terminals to be bi-directionally switched, under program control, between two Series/1s
- Support for the IBM 4968 Autoload Streaming Magnetic Tape Unit enables quickly backing up the data from a large-capacity disk
- Support for the integrated 30-megabyte disk unit in the IBM 4954-30D and 4956-30D processors and in the 4965-30D storage and I/O expansion unit
- Support for the IBM 4980 Display Station
- Support for the IBM 5219, 5224, and 5225 Printers
- Support for additional command language facility commands
- Support for additional operator commands
- Terminal backup capability of the Multiple Terminal Manager in case of processor failure
- Asynchronous processing capability of the Indexed Access Method improves the performance of application programs

Changes to this Book

- Clarify that magnetic tape units can not be distributed devices. A program using a magnetic tape unit must run at the Series/1 where the tape unit is attached.
- Removal of the restriction that the 3101 Display Terminal can not be a distributed device. All supported terminals can now be distributed devices. A program running on one Series/1 in a multiprocessor system can use terminals that are attached to other Series/1s in the system.
- PASSTHRU service of the Multiple Terminal Manager can now use any of its full-screen formatted terminals

Summary of Amendments

Summary of Amendments
for *Concepts and Facilities*, GC34-0471-0
as updated by GN34-0840
Realtime Programming System Version 6 Modification Level 1

Additions to this Book

- The licensed program, Communications Manager for the Series/1, is supported
- The Multiple Terminal Manager has a programming interface to the Communications Manager
- The X.25/HDLC Communications Support licensed program functionally replaces the Packet Network Support programming RPQ
- The Remote Manager licensed program is supported
- Task storage records are available for supervisor-state tasks. The system maps the task storage record into the same area of logical storage each time it dispatches the task.
- You can develop a system that supports an additional terminal, or a terminal emulator program (that is, a virtual terminal), as the operator console or as a display device
- A full-screen editor (FSEDIT) is available for examining and changing data in hexadecimal format
- Operations on X.21 circuit switched public data networks are supported in binary synchronous and SNA/SDLC modes in compliance with the CCITT X.21 recommendation
- You can use the ICDEF command to specify internode communications control block allocations when using the command language facility
- IBM-supplied subroutines provide an interface between Cobol, Fortran, or PL/I, and the operating system services of loadable external modules and global queues
- IBM-supplied subroutines provide an interface between PL/I and the operating system storage services
- In text editor full-screen mode you can tailor the program function keys
- The 4967 disk is supported

Summary of Amendments

Version 6 of the IBM Series/1 Realtime Programming System is the latest version of the system. This summary of amendments is for users of previous versions of the Realtime Programming System who might want to know how this book, *IBM Series/1 Realtime Programming System Version 6: Concepts and Facilities*, reflects system function that is new, changed, or deleted for Version 6.

New Function

New functions of the Realtime Programming System for Version 6 are listed below.

Multiprocessor System Support

Information has been added to document support for the Series/1 Realtime Programming System as a multiprocessor system with several processors linked together in a configuration that makes possible a level of fault tolerance and incremental system growth.

Loadable External Module Facility Documentation

Information has been added to document loadable external modules, which are program modules that can be loaded and unloaded anywhere in the storage of your partition while your task set is executing.

Duplex Volume Documentation

Information has been added to document duplex volumes.

Query Licensed Program Documentation

Information has been added to document a new licensed program called Query, a program for retrieving and updating data in sequential or indexed files.

Interactive System Utilities Update

Information has been added to document program function key tailoring capability for program function (PF) keys 1 to 5, and output scrolling capability using the REPORT system utility.

Two-Channel Switch Support

Information has been added to document support of the Series/1 two-channel switch (hardware #7900) for use in recovering from processor failures.

New Service Aids Documentation

Material on the following new service aids has been added:

- Error log types
- Online device tests
- Automatic time stamping of all system messages

New Operator Commands

Information has been added to reflect new operator commands.

New Tape Utilities

Information has been added to document tape utilities as part of the base operating system.

New Stand-Alone Utility

Information has been added to document a new stand-alone utility, REPT, which produces a formatted report of a task set reference table.

New Terminal Controller Commands

Information has been added to document new terminal controller commands, LOGOFF and LOGON CONSOLE.

Changed Functions

Changed functions of the Realtime Programming System for Version 6 are listed below.

Advanced Remote Job Entry Update

Information has been added to document miscellaneous enhancements to the Advanced Remote Job Entry program product.

Packet Network Support

Information has been added to document miscellaneous enhancements to the Packet Network Support programming RPQ.

Multiple Terminal Manager Update

Information has been added to document miscellaneous enhancements to the Multiple Terminal Manager program product.

Command Language Facility Update

Information has been added to document enhancements to the command language facility, most notable of which is the addition of a set of menus (and accompanying help screens) to facilitate use of system services.

System/370 Host Assembler Programming RPQ Documentation

Information has been added to document a programming RPQ called the System/370 Host Assembler, a language translator that translates symbolic instructions into Series/1 machine language instructions, assigns storage locations, and performs auxiliary functions necessary to produce executable machine language programs.

Indexed Access Method Update

Information has been added to document miscellaneous enhancements to the Indexed Access Method program product.

SNA Extended Update

Information has been added to document SNA extended as part of the SNA support in the base supervisor.

Deleted Function

Deleted functions of the Realtime Programming System for Version 6 are listed below.

- Prebinding task sets
- Rollout/rollin partitions
- Message buffering
- 4952 processor
- 5250 terminal

Chapter 1. Overview of the Realtime Programming System

The Realtime Programming System is an operating system for the IBM Series/1 processor. IBM also provides other licensed programs for use with the Realtime Programming System. These include high-level languages (such as PL/I, Fortran, Cobol, and Pascal), program development tools, and data communications facilities.

Contents of this chapter:

Operating System Characteristics

General attributes of the Realtime Programming System are shown in Figure 1-1 .

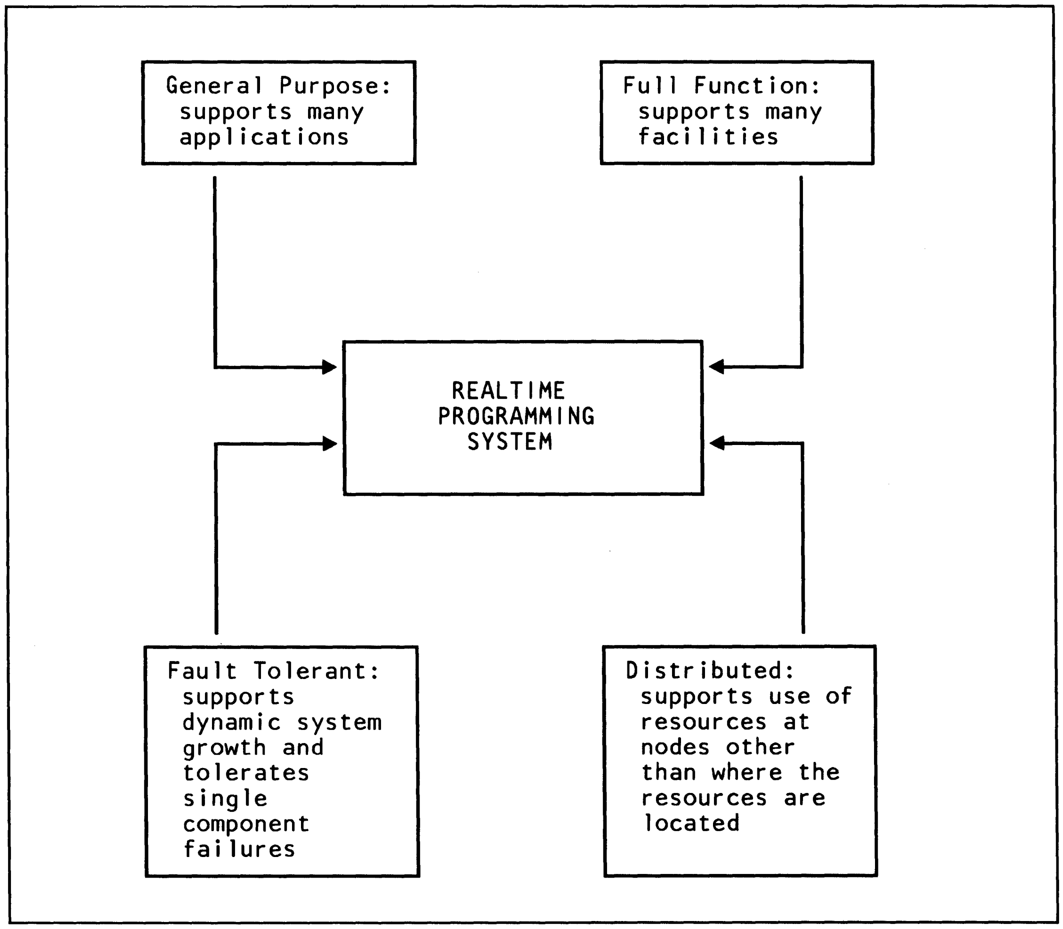

General Purpose: supports many applications
Full Function: supports many facilities
REALTIME PROGRAMMING SYSTEM
Fault Tolerant: supports dynamic system growth and tolerates single component failures
Distributed: supports use of resources at nodes other than where the resources are located

Figure 1-1. Attributes of the Realtime Programming System

- **General purpose** means the operating system supports many types of processing (see "The Types of Work the Realtime Programming System Can Do" on page 1-3)
- **Full function** means it provides the necessary facilities to write many different kinds of data processing applications
- **Fault tolerant** means the system is designed so that it continues running if a single Series/1 in a system of Series/1s fails
- **Distributed** means that a program running on one Series/1 in a multiprocessor system can use resources that are attached to other Series/1s in the system. Such resources include disks, diskettes, printers, and terminals.

Fault Tolerance, Incremental Growth, Multiprocessor Operation

The Realtime Programming System provides:

- Fault tolerance

- Incremental growth
- Multiprocessor operation

Once an enterprise decides to use computers, it becomes dependent on its computer systems for its daily operation. These systems must be able to handle failures while still providing service. Systems that can handle the failure of a single hardware component are **fault tolerant**.

Also, as the enterprise grows or changes, computing needs increase. **Incremental growth** enables you to add more computing resources without disrupting the system's operation.

The Version 6 Realtime Programming System promotes a high degree of fault tolerance and incremental growth by enabling:

- A system to grow or shrink in size without taking the system down
- A user or system program to use many devices (such as disks) no matter where the devices are attached in the system
- A second copy of a file to be kept automatically so processing can continue if one copy fails
- The home **node** supervisor functions to be backed up. (A node is a processor in a multiprocessor system together with its associated programs and devices.)
- More than one Series/1 to be under the control of a single operating system
- A set of common terminals can be bi-directionally switched, under program control, between two processors. This dynamic reconfiguration can recover these common terminals from a failing processor, and return them when appropriate.

A **multiprocessor system** means from two to sixteen Series/1s are connected by two Local Communications Controllers. Each Local Communications Controller forms a high-speed data link between nodes.

The Types of Work the Realtime Programming System Can Do

Using the Realtime Programming System, you can do the following types of data processing:

- Transaction processing
- Commercial data processing
- Data acquisition and control
- Message transmission and message concentration
- Distributed data processing
- Data communications and data exchange
- Batch processing

Transaction Processing

A transaction processing application can consist of a set of programs that can use data files and display formats on a terminal screen. A single processing application can request input from end users by means of one or more displays. These are called transaction interactions. For example, the user displays and updates the operational data of a business enterprise. In most cases, the end user has little or no knowledge of data processing. Transaction processing applications can be written in assembler, Cobol, Fortran, Pascal, or PL/I.

Commercial Data Processing

A commercial data processing application involves such things as sorting or merging files, performing arithmetic calculations and logical operations, generating printed reports, and updating files. The application can be written in a high-level language such as Cobol, Fortran, PL/I, or Pascal. The operating system supports these languages.

Data Acquisition and Control

In data acquisition and control programs, the system must react to external stimuli or sensors quickly. Programs are scheduled and run in response to these external stimuli. The operating system:

- Handles data from sensors (analog and digital input and output)
- Manages external interruptions (process interrupts)
- Schedules and runs programs

You can write data acquisition and control programs using PL/I, Fortran, or Series/1 assembler language.

Message Transmission and Message Concentration

A message is any unit of information to be transmitted. A message can be a single sales transaction or an entire file of data. In programs that transmit messages, a message (unit of information) received by a central system from one terminal is sent to one or more systems or terminals. These programs require that the system handle communications lines, different types of terminals, and various communications protocols.

Programs that perform message concentration collect and concentrate messages in one location and then forward them to another location. Data flow control routines select alternate paths for sending the messages to ensure continuous operation. There is often a need for Series/1 systems to connect to other machines.

Message handling facilities are part of the Multiple Terminal Manager and Communications Manager licensed programs.

Distributed Data Processing

A distributed data processing application is one in which some or all of the processing, storage, control, and I/O functions are situated in different places and are connected by transmission facilities.

Data Communications and Data Exchange

Data communications facilities allow data to be exchanged with other computer systems or devices. The form in which the data is sent must allow the data to be used on the receiving system or device. Data to be exchanged is transmitted to another system **electronically**, over communications lines, or **physically**, by recording the data on a removable storage medium that can be used on the receiving device or system.

Data can be exchanged electronically between a Series/1 system and another system that supports at least one of the following communications protocols:

- Asynchronous (start/stop) communications protocol
- Binary synchronous communications protocol
- Synchronous data link control (SDLC) protocol in a Systems Network Architecture (SNA) network

These protocols can be used to connect many different types of devices. You should review the following manuals dealing with Realtime Programming System communications protocols for more specific information:

- *Binary Synchronous and Start-Stop Communications Programming Guide*
- *Systems Network Architecture Support Programming Guide*

Data can be physically exchanged between a Series/1 system and other systems, provided the other systems can use any of the following media:

- 9-track magnetic tape (non-labeled tapes and IBM standard labeled tapes compatible with System/370 DOS/VS)
- IBM basic exchange diskettes
- IBM type E exchange diskettes
- IBM type H exchange diskettes

Batch Processing

Long running, low priority programs that do not require interaction with a terminal are often batched. Batch processing means a set of programs are stacked or batched together and run in first-in-first-out or priority order.

The job stream processor is a Series/1 batch processing facility.

Uniprocessor and Multiprocessor Systems

A Version 6 **system** contains one or more Series/1s under the control of the Realtime Programming System. The operating system recognizes two system types. A **uniprocessor** system consists of one Series/1. A **multiprocessor** system consists of two or more Series/1s (maximum of 16) connected by two Local Communications Controller rings.

Each Series/1 is a **node** in a multiprocessor system. From a program's point of view, a **local node** is the Series/1 in the system where the program is running. A **remote node** is any other node in the system. The **home node** is the node from which all the nodes in the system were first loaded.

A multiprocessor system has **distributed** devices. These devices include disks, diskettes, printers, and terminals. Any program in the system can use these devices as long as the Series/1 with the devices is online.

Uniprocessor systems are typically used either to execute applications or to develop new ones. This book uses the terms "production system" and "development system" to describe these uses. Uniprocessor systems are not usually powerful enough to support both activities simultaneously. In contrast, a multiprocessor system can be both a development and a production system at the same time because of its greater power. The term "standard system" refers to the pre-built operating system you receive from IBM.

Uniprocessor System

A uniprocessor system consists of one Series/1. Past users of the Realtime Programming System can use the operating system as they have in the past if they do not need a multiprocessor system. The *System Planning Guide* discusses differences between Realtime Programming System Version 6 and earlier versions of the Realtime Programming System.

Multiprocessor System

A multiprocessor system consists of two to sixteen Series/1 processors attached by one or two Local Communications Controllers.[1] Each Local Communications Controller provides a high speed data link between nodes. The operating system controls all Series/1 systems, the two Local Communications Controllers, and devices.

In Realtime Programming System Version 6:

- The supervisor controls the flow of data and commands over the Local Communications Controller
- Task sets can be queued to be run at another node
- Resources such as global queues or global resources are known throughout the system. The system uses the home node to synchronize the use of resources in the system.
- Disks, diskettes, printers, and terminals can be used by programs running anywhere in the system. They need not be at the node where the program using the devices is running.[2]

Incremental Growth

In a multiprocessor system, you can add or remove nodes while the system is running. If you take a node out of the system to add or remove devices, remaining nodes in the system continue running. A program called the **configurator** helps you to grow or shrink the size of a multiprocessor system.

Configurator

The configurator is the program you use to set up, install, or change the makeup of a multiprocessor system. You start the program using a command language facility command.

1 To provide a high degree of fault tolerance, each Series/1 must be attached to two Local Communications Controllers so that processing can continue if one fails.

2 Not all devices can be attached to a Series/1 different than the one using the device. For example, a program using sensor I/O, magnetic tape, or communications devices must run at the node where the device is attached.

(The command language facility is an interactive set of menus and commands for programming and managing files.) Once you enter the command to start the configurator, menus appear on which you enter the needed information to configure your system. The configurator uses the facilities of the Series/1 hardware to identify the I/O devices attached to a node and to enable you to add items to the system.

System Backup and Recovery

To ensure fault tolerance in a multiprocessor system, Version 6 provides backup procedures for data and certain hardware in the system. For critical data, you can designate that the disk volume containing the data be duplexed.

For system backup, Version 6 requires two Local Communications Controller rings in a multiprocessor system. One ring serves as a backup for the other ring.

You can select a node to function as a backup for the home node. If you do not, the system chooses one of the remaining nodes as a backup node.

You can use the manual Two-Channel Switch feature (#7900) of the 4959 I/O Expansion Unit to electronically switch I/O devices from a failing node to a backup node. Upon processor failure a Switchover Command File, containing operator commands, is opened and executed in the backup node. The commands typically start the switched devices and the applications that service them, but any operator commands allowed in UICINPUT are acceptable. Once the failed node is restored, the operator must manually reset the switch to move the switched I/O devices back to their original state.

You can use the Programmable Two-Channel Switch feature (#7777) of the 4959 I/O Expansion Unit to electronically switch common terminals from a failing node to a backup node. A set of common terminals can be bi-directionally switched, under program control, between two processors. This dynamic reconfiguration can recover these common terminals from a failing processor, and return them when appropriate. The Programmable Two-Channel Switch can also be manually operated.

The Two-Channel Switch and the Programmable Two-Channel Switch are mutually exclusive; that is, they cannot both be installed on the same pair of nodes.

The Standard System

The **standard system** is a pre-built operating system included in the set of diskettes that IBM sends when you order a uniprocessor or multiprocessor system. It is the first operating system you install on a Series/1 processor. The standard system must be installed on the processor of each node in a multiprocessor system.

The standard system enables you to begin using the Series/1 quickly and easily. Using the standard system, you can add IBM licensed programs, devices, or user-written programs without extensive planning or delay.

If you install the Program Preparation Subsystem under the standard system, you have an operating system that provides most of the tools programmers need to write and test programs.

Development and Production Systems

Version 6 systems can support either a program development or production environment, or both.

Program development involves using languages, libraries, and other tools to write and test programs. A system that supports this activity is called a **development system.**

After a program is fully tested, it is used in production mode. A system customized to execute application programs is called a **production system.**

Development System

IBM supplies a licensed program with the tools a programmer needs to write and test programs. This licensed program is the **Program Preparation Subsystem**. Program development components of the Program Preparation Subsystem are shown in Figure 1-2.

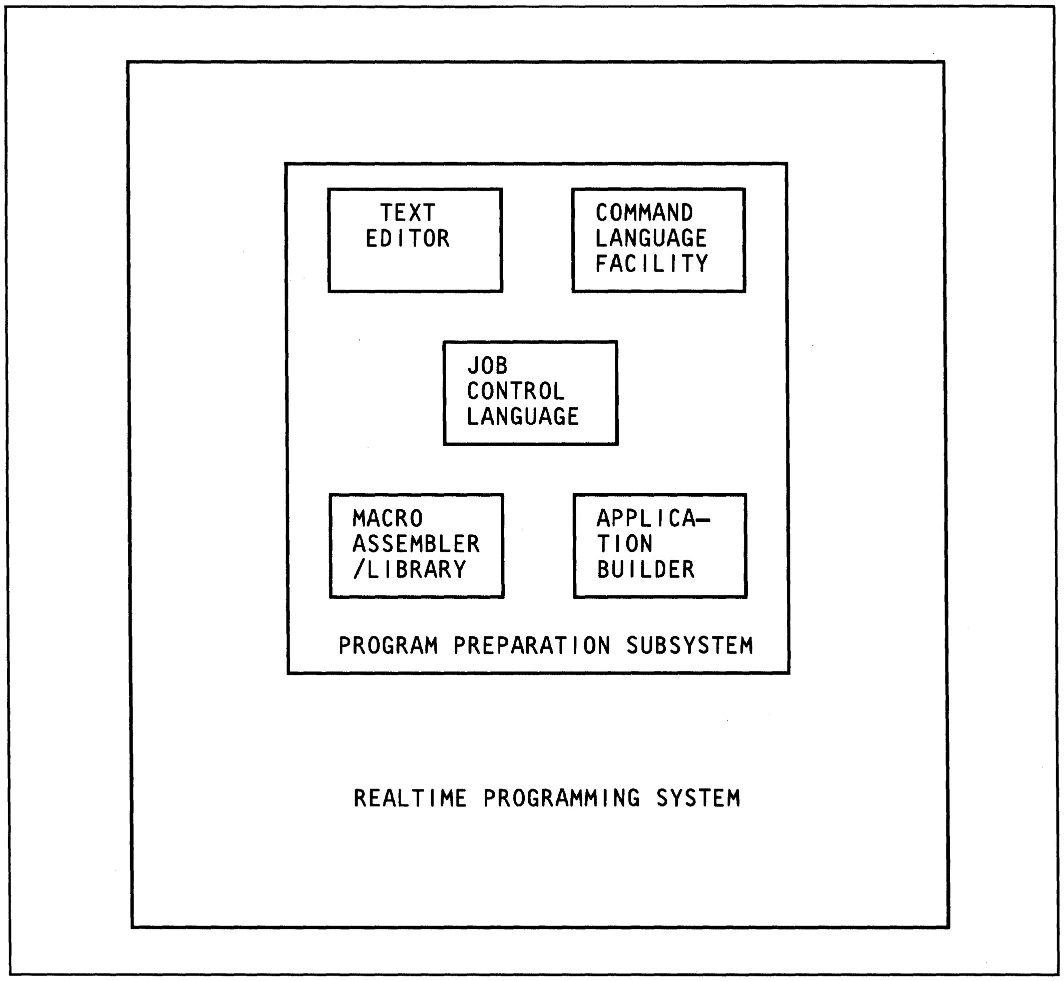

TEXT EDITOR
COMMAND LANGUAGE FACILITY
JOB CONTROL LANGUAGE
MACRO ASSEMBLER /LIBRARY
APPLICA- TION BUILDER
PROGRAM PREPARATION SUBSYSTEM
REALTIME PROGRAMMING SYSTEM

Figure 1-2. Development Components of the Program Preparation Subsystem

The Program Preparation Subsystem includes:

- A full-screen text editor with commands to edit and update source program files and data files
- Interactive menus and commands of the command language facility for writing and testing programs
- A batch job control language to create and execute long compilations or assemblies that do not require interaction at a terminal. The job stream processor provides this command language.
- A macro assembler and macro library for writing programs in Series/1 assembler language. The assembler language includes structured programming macros. Routines in the macro library interface to system functions.
- An application builder facility with statements that combine compiler or assembler output into programs that run under operating system control

All components of the Program Preparation Subsystem run under the control of the Realtime Programming System.

In addition to the Program Preparation Subsystem, four high-level programming languages are provided (as separate products):

- Cobol
- Fortran
- Pascal
- PL/I

Production System

When the writing and testing cycle of a program is complete, you install the program on the system. A program serving its end users runs in production mode. Throughout this book, the term "production system" describes a Realtime Programming System that you custom build to support a production environment. In some cases, the IBM-supplied standard system is suitable for a production environment.

Changing the Standard System

You can change (customize) the IBM-supplied standard system to support a unique environment. For example, hardware features, devices, and licensed programs (such as language compilers) that are supported on a development system might not be needed on a production system. Conversely, devices not needed on a development system may be necessary for a production system.

You can also change the standard system by changing special system files (the initial program load (IPL) input file or the user input command (UIC) file). The configurator aids you in this process. These files and the configurator are discussed in the section entitled "Customizing Your Operating System" on page 2-41.

Using the SYSGEN Program

You use the SYSGEN program to custom build a Realtime Programming System to include only those components you need. The SYSGEN program is a part of the Program Preparation Subsystem.

You can also customize the Realtime Programming System to run on a uniprocessor system with no fixed disk device; a diskette can be the system-resident device.

Major Software Components of the Realtime Programming System

This section presents you with a brief overview of the functions of the operating system, and introduces several terms that are used throughout this book.

Realtime Programming System components are grouped into four categories:

- The control program
- Development tools
- Application aids
- Data communications tools

The Control Program

The control program consists of the:

- Supervisor
- Data management routines
- Menus and commands of the command language facility
- System utilities
- Tape utilities
- Stand-alone utilities
- Operator interface
- Data communications support

The Indexed Access Method is a separately priced program that extends the input/output support of the control program.

Supervisor

The **supervisor** manages one or more Series/1 processors within a single node. It handles functions such as multiprogramming, multitasking, error processing, storage management, and timer management. Supervisor functions are discussed in more detail in Chapter 2, "Operating System Facilities" and in the *Supervisor Services Programming Guide*.

Data Management Routines

Data management routines manage input and output to files and devices. The routines provide functions that:

- Allocate and provide access to data files
- Provide access to disks, diskettes, magnetic tape, sensor input/output devices, printers, and several types of terminals

System I/O functions are discussed in more detail in Chapter 2, "Operating System Facilities" and in the *Data Management Programming Guide.*

Menus and Commands of the Command Language Facility

Interactive menus and commands of the command language facility are included in the control program for programming and for managing files. The facility can also be used by the support programmer to install other Realtime Programming System based licensed programs such as PL/I, Fortran, Cobol, Pascal, and the Program Preparation Subsystem.

Once started, the facility creates an interactive terminal session for each of its users.

The facility includes interactive menus and commands for performing various functions. Help screens assist you in using the set of menus. You enter the commands online at the terminal. The facility also includes a procedural language called S1/EXEC for writing your own commands or menus.

The command language facility is discussed in more detail in Chapter 5, "Development Tools and Application Aids" and in the *Command Language Facility User's Guide.*

System Utilities

The **system utilities** are a set of interactive routines that can be used to manage data stored in system format files on disk or diskette. (System utilities can also do some processing on non-system format diskettes.) The utilities program can be started under command language facility control.

The system utilities are discussed in more detail in Chapter 5, "Development Tools and Application Aids" and in the *Utilities Reference* manual.

Tape Utilities

The **tape utilities** are a set of commands for backing up data from a disk to a tape (and tape to disk), copying data from tape on one drive to tape on another drive, building a tape data set definition, and initializing or exercising a tape.

The tape utilities are discussed in more detail in Chapter 5, "Development Tools and Application Aids" and in the *Utilities Reference* manual.

Stand-Alone Utilities

The **stand-alone utilities** are a set of commands and programs for formatting disks, saving and restoring volumes or devices, loading and starting a supervisor, dumping processor storage to diskette, and producing directory reports.

The stand-alone utilities are discussed in more detail in Chapter 5, "Development Tools and Application Aids" and in the *Utilities Reference* manual.

Operator Interface

The **operator interface** consists of a set of commands that enable an operator to start and stop programs, control I/O devices, monitor system status, and make backup copies of files.

The operator commands are discussed in more detail in Chapter 4, "User Interfaces to the System" and in the *Operation Guide and Reference* manual.

Data Communications Support

Data communications support allows communications with remote locations and consists of those portions of the control program that provide:

- Binary synchronous communications
- Asynchronous (start/stop) communications
- Series/1 support for the Systems Network Architecture (SNA)

Systems Network Architecture support for a Series/1 is a subset of the total IBM Systems Network Architecture support.

Data communications support is discussed in more detail in Chapter 3, "Data Communications and Network Support" and in the following books:

- *Binary Synchronous and Start-Stop Communications Programming Guide*
- *Systems Network Architecture Support Programming Guide*

Indexed Access Method

The Indexed Access Method program builds and maintains indexed files and enables access, by symbolic key, to records in an indexed file.

The Indexed Access Method is discussed in more detail in Chapter 5, "Development Tools and Application Aids" and in the *Indexed Access Method User's Guide.*

Development Tools

Four high-level languages (PL/I, Cobol, Fortran, and Pascal) and a group of program development facilities called the Program Preparation Subsystem provide the tools used on a development system.

PL/I

The following licensed programs are required for program development in PL/I:

- PL/I Compiler and Resident Library
- PL/I Transient Library

The transient library is also necessary to execute programs produced by the PL/I compiler.

Cobol

The following licensed programs are required for program development in Cobol:

- Cobol Compiler and Resident Library
- Cobol Transient Library

The transient library is also necessary to execute programs produced by the Cobol compiler.

Fortran

The following licensed programs are required for program development in Fortran:

- Fortran Compiler and Object Support Library
- Mathematical and Functional Subroutine Library

Fortran requires the Mathematical and Functional Subroutine Library to build and execute programs produced by the Fortran compiler. Fortran also requires either the floating point hardware feature or floating-point emulation support in the system.

An additional licensed program, the Fortran Realtime Subroutine Library, is needed for applications that use sensor I/O, realtime functions, or system multitasking functions.

Pascal

The Pascal Compiler and Object Support Library licensed program is required for program development in Pascal.

Program Preparation Subsystem

A licensed program, called the Program Preparation Subsystem, provides common tools for program development. Features of the Program Preparation Subsystem include:

- A full-screen text editor with commands to edit and update source program files and data files
- A batch job control language to create and run long compilations or assemblies that do not require interaction at a terminal. The job stream processor provides this command language.
- A macro assembler and macro library for writing programs in Series/1 assembler language. Macros are provided to assist in structured programming and interface to system functions.
- An application builder facility to combine compiler or assembler output into programs that execute under the operating system
- A system generation program (SYSGEN) to custom build a supervisor
- Interactive menus and commands of the command language facility for writing and testing programs

The Program Preparation Subsystem runs under the Realtime Programming System.

Application Aids

Application aids support functions that are frequently needed on both development and production systems. These optional separately-priced licensed programs aid you in writing applications.

Multiple Terminal Manager

The Multiple Terminal Manager program supports transaction processing. The manager supplies a set of utilities that aid you in creating a transaction processing application that uses multiple terminals. The application can be written in PL/I, Cobol, Fortran, Pascal, or assembler language.

The Multiple Terminal Manager also supports remote transaction processing. This support allows a Series/1 processor and specified terminals to appear to a host system as one or more 3270 Information Display Systems while local transaction applications use other terminals attached to the Series/1 processor.

The manager also allows you to access data on other Series/1 processors by using a programming interface to the Communications Manager. This support is discussed in the section entitled "Multiple Terminal Manager Bridge Service" on page 3-18.

Query

Query is a program that enables you to get information from Series/1 files without writing programs. Query can supply timely, appropriate, and concise information.

Using Query, you can look at data in an Indexed Access Method file or a sequential file, or you can update, delete, or insert data in an Indexed Access Method file.

As a tool for information analysis, Query can turn your system into a management information system. Query provides menus for you to select the functions you wish to do.

Query must be run under control of the Multiple Terminal Manager in order to set up file profiles. You can also call Query from a Cobol or assembler language program.

Mathematical and Functional Subroutine Library

The Mathematical and Functional Subroutine Library routines are available to assembler language and Fortran programmers.

Sort/Merge
The Sort/Merge licensed program sorts and merges records from up to eight input files into one output file in either ascending or descending order.

Job Stream Processor
The Job Stream Processor Programming RPQ is a batch processing facility for executing a collection of background[3] programs on production systems. This programming RPQ is the same as the Job Stream Processor component of the Program Preparation Subsystem.

Data Communications Tools

Data communications tools provide functions that extend the base communications support and the operating system support for Systems Network Architecture (SNA) networks. These optional separately-priced licensed programs are discussed below.

Communications Manager
The Communications Manager for the Series/1 manages the flow of messages in a network between Series/1s, between Series/1s and host computers, and between the Series/1 and a variety of input and output devices. The Communications Manager provides operator commands to allow online control of the network and to display information about activity within the network. Message management features of the Communications Manager include line concentration, message queuing, message warm start, the ability to obtain messages from any supported source (computer, device, or application program) and the ability to deliver messages to any supported destination on a priority basis.

System/370 Channel Attach Program
The System/370 Channel Attach Program allows a Series/1 application to communicate with an application executing on a host System/370, 30xx, or 43xx system. The Channel Attach Program uses the Series/1-to-System/370 Channel Attachment hardware. The host system must be using OS/VS1, OS/VS2 (SVS or MVS), or DOS/VSE with the Basic Telecommunications Access Method (BTAM).

[3] Programs that execute in background mode are usually long-executing, low-priority programs that do not require interaction with a terminal. These programs are queued (batched) and executed one after another. Programs that execute interactively at a terminal are said to run in foreground mode.

Advanced Remote Job Entry

The Advanced Remote Job Entry program supports the use of a Series/1 as a remote job entry work station using either Systems Network Architecture (SNA) or multi-leaving binary synchronous communications.

Advanced Remote Job Entry:

- Supports multi-leaving remote job entry support for binary synchronous communications using EXIO BSC support
- Supports multiple logical unit Systems Network Architecture (SNA) support for Synchronous Data Link Control (SDLC) using SNA and SNA extended
- Has commands that are designed to be easily used and that are identical for remote job entry operation
- Supports **unattended operation** by having remote job entry commands on disk or diskette. Support for dynamic punch file allocation and delayed activation is also provided.
- Has full function console support with status reporting and journaling, data decompression, and printer form support

SNA Remote Management Utility

The SNA Remote Management Utility Programming RPQ allows you to communicate between a host computer and a remotely located Series/1. The utility allows you to perform various file maintenance, system maintenance, and system functions without having an operator at the remote Series/1. The SNA Remote Management Utility executing on a Series/1 processor gives the host system access to Series/1 Remote Management Utility functions.

4987 Programmable Communications Subsystem Preparation Facility

The 4987 Programmable Communications Subsystem Preparation Facility is used to support the generation of controller storage image programs for the Programmable Communications Subsystem. The program provides a macro library for assemblies with the Program Preparation Subsystem.

4987 Programmable Communications Subsystem Extended Execution Support

The 4987 Programmable Communications Subsystem Extended Execution Support is used to write application programs that communicate with devices that are attached via the 4987 Programmable Communications Subsystem. Two levels of user interface are supported; READ/WRITE and EXIO.

X.25/HDLC Communications Support

The X.25/HDLC Communications Support licensed program enables a Series/1 to attach to X.25 packet switching networks. You can also use the X.25/HDLC Communications Support to communicate with terminals or processors in a network that uses high-level data link control (HDLC) procedures.

Remote Manager

The Remote Manager licensed program enables Series/1 networks to be managed and operated through the communications and systems management programs available on IBM host processors (System/370, 30XX, and 43XX). The Remote Manager on each Series/1 in the network supports centralized control and problem determination using the following host programs:

- Network Communications Control Facility
- Distributed Systems Executive
- Host Command Facility
- Network Problem Determination Application

Summary

The Realtime Programming System and its supporting programs provide a wide range of functions. The operating system is general enough to handle traditional commercial data processing, yet is responsive enough to handle many transaction processing, interactive, or process control applications.

Chapter 2. Operating System Facilities

This chapter highlights general operating system concepts and discusses Realtime Programming System facilities that provide various means of solving data processing problems. The term **system service**, as used in this book, describes a programming interface to a system facility that consists of one or more interactive commands, assembler language macros, or interactive applications.

Contents of this chapter:

Keeping the System Productive

Effective utilization of a system implies that system resources must be kept fully utilized in a productive fashion. A system is kept productive by:

- Keeping the processors busy
- Sharing processing units among tasks
- Using I/O to balance system performance

Keeping the Processors Busy

A Series/1 processor is equipped with a single processing unit and therefore executes a single set of program instructions at a time. Time and processor resources are wasted if the program in control of a processing unit is waiting for an I/O operation, for a resource to become available, or for human intervention before execution proceeds.

Multiprocessing

Multiprocessing means running more than one program at the same time. Because a multiprocessor system has multiple processors, more than one application program can execute, at different processors, at the same time.

Multiprogramming

An idle processing unit represents lost work if another program is awaiting execution. The procedure for finding ready programs in processor storage and for starting their execution is called **multiprogramming**. Multiprogramming gets its name from the idea of multiple programs sharing a processing unit's execution cycles.

Multiprogramming allows more than one program to reside in processor storage and appear to be executing at the same time. The operating system chooses which program is to receive control of the processing unit. The processor allows instruction execution and I/O operations to overlap. While one program is waiting for I/O to complete, another program is permitted to execute. Multiprogramming works because the operating system is able to recognize and respond to interruptions in processing.

Multitasking

If an application is divided into independent units of work, there are more instances in which the supervisor can find sequences of instructions that can be given control of the processing unit to do work. This process is exactly what occurs within the operating system where a large number of independent units of work are created. The independent units of work are called **tasks**, and the supervisor allocates machine cycles to these tasks. This sharing process is called **multitasking**.

Tasks and Task Sets

A program's execution structure is divided into one or more tasks. The resulting structure is called a **task set**.[1] Any program that executes under the Realtime Programming System contains at least one task called the **primary task**. A program with multiple tasks has one primary task. All other tasks are called **secondary tasks**. All of the tasks for a single program execute on the same node that executes the primary task. The supervisor shares the processing unit among the tasks in your program as well as those in other programs executing in the processor.

Program Modules Within a Task

You write modular programs by dividing a task into **program modules** that execute synchronously. A task contains one main (primary) program module — the module that receives control when a task begins executing. Other program modules within a task are **subprograms** (secondary programs). Subprograms are executed only if they are called by the main program or by a subprogram that is executing. When the main program module completes execution, the task containing the program module terminates.

How the Supervisor Shares a Processing Unit Among Tasks

The supervisor shares a processing unit among tasks by using a component called the dispatcher, and events.

The Dispatcher

In order to distribute machine cycles among competing tasks in an effective manner, the supervisor recognizes when a task is temporarily unable to proceed. Also, the supervisor recognizes when the required conditions have been met so that a suspended task can resume execution. The **dispatcher**, a component of the supervisor that runs in every node, decides when the processing unit for that node is allocated to a particular task.

Task Execution

All tasks running on a given processor compete for processor execution time. If no constraints exist, tasks execute and terminate asynchronously. Events allow the execution of a task to be synchronized with the execution of other tasks or with operating system tasks. An **event** is an occurrence that is significant to a task (such as the completion of I/O). Events might be necessary if tasks need to communicate information, if there is a point in one task where further execution is dependent upon an external occurrence, or if another task is dependent on your task (two or more tasks must cooperate).

[1] A task set is a means of packaging together the parts of an executable program. These parts include the program instructions, work areas, and optional data areas. The task set consists of related tasks that perform an application. The terms **task set** and **program** are used interchangeably in this book.

Factors That Determine Task Execution

Whether or not a task executes depends on many factors including:

- The status of input or output operations that are executed on behalf of the task
- The recognition of the occurrence of events in a task's execution. An event can consist of an interruption from an external sensor, the expiration of a time interval, or in general, the arrival of a cooperating task at a specified point during execution. In the last case, a task notifies another task that it is awaiting the occurrence of an event. The task then suspends itself (waits) until the other task indicates (posts) that the event has occurred.
- The availability of a serially reusable resource such as a program segment or data area. A resource is serially reusable during execution if only one cooperating task can use the resource at a time. You use system facilities to define the resource as serially reusable. The supervisor grants control of serially reusable resources to one task at a time. If a task requests a resource that is already being used, it can suspend itself until the resource is no longer being used. Alternatively, the task can continue execution and try again later.
- Task **dispatching priority**. You give a task a dispatching priority to:
  - Promote system throughput. For example, you should give a task which does not use its processor heavily, but which does a lot of I/O, a high priority.
  - Make sure the processor is available to tasks with critical response or service requirements (such as process interrupt servicing for process control)
  - Assure cause and effect processing. For example, you should give a problem state program that uses a device a lower priority than the system task that manages the device.

  You assign a task's dispatching priority. The dispatcher uses the priority of a task to decide which task gains control of the processing unit when several tasks are ready to execute.

Using I/O to Balance System Performance

Each Series/1 processor is equipped with an I/O channel and I/O devices containing dedicated microprocessors. The channel allows input and output operations to proceed without the assistance of the processing unit. This independent processing allows the operating system to support both asynchronous and synchronous I/O.

Asynchronous I/O

A program that uses **asynchronous I/O** performs its I/O independently of its execution. You use asynchronous I/O if your program can continue executing without waiting for an I/O operation to complete.

Asynchronous I/O is accomplished by specifying an event on your I/O request. Your program continues execution until the program reaches a point where it must wait for the event to occur before proceeding. Asynchronous I/O allows a program to overlap execution with I/O operations.

Synchronous I/O

A program that uses **synchronous I/O** must wait for an I/O operation to complete before continuing execution. The operating system puts a program using synchronous I/O into a wait state while the I/O is being done. When the I/O is complete, the operating system resumes the program's execution.

Remote I/O and Load Balancing

An application may require more I/O devices than can be attached to, or adequately serviced by, a single processor. In a multiprocessor system, programs in any node can refer to certain devices attached to any other node. The locations of disks, diskettes, printers, and terminals are transparent to your application programs. I/O requests are routed to the node that has the device(s) physically attached to it. Only programs using sensor I/O, magnetic tape, or communications devices must run at the node where the device is attached.

Utilizing processors at several nodes for device or file servicing is particularly beneficial if you are using the Indexed Access Method.

Spooling

The operating system provides **spooling** services that allow you to send output to a slow speed device without having the slow speed device limit the execution speed of your program. The operating system first writes the output data to a disk file at high speed and later writes the data to its final destination. Spooling allows your program to print without having to wait for the data to be transmitted to the printer. Your program then completes its execution so that other work can be started while the system completes the transmission. The final output occurs asynchronously with your program's execution.

Spooling services manage contention for a printer when several programs attempt to use the printer at the same time. When directing spooled output to a printer you may specify an output class. The operating system, under the operator's control, decides what spooled output is to be printed based on the class designation.

The operating system allows you to select different output forms on which output is to be printed. You can also request multiple copies of spooled output. The operating system interacts with the operator when non-standard forms are requested so that the proper output forms can be mounted before printing proceeds.

Spooling is discussed in more detail in the *Operation Guide and Reference* manual.

Optimizing the Use of Processor Storage

Processor storage refers to the hardware that retains instructions and data within each processor. Processor storage is a fundamental resource of your system. The demand for processor storage forces the operating system to manage the storage resource as effectively as it can. You control how the operating system manages processor storage.

Logical storage is the conceptual storage layout of a processor which represents address spaces. The operating system allows programs to use the addressing scheme of this conceptual storage layout to access processor storage. To do this, the operating system manages a group of registers called segmentation registers.

A stack of segmentation registers is identified by an address key. The logical storage defined by an address key is called an address space. The storage address relocation translator in each processor uses the values in the segmentation registers to convert logical storage (also called primary storage) addresses into processor storage addresses.

How Storage is Managed

Management of primary storage under the Realtime Programming System is done on a node by node basis and starts with the concept of **partitions**.

Partitions

An example of partition arrangement is shown in Figure 2-1 .

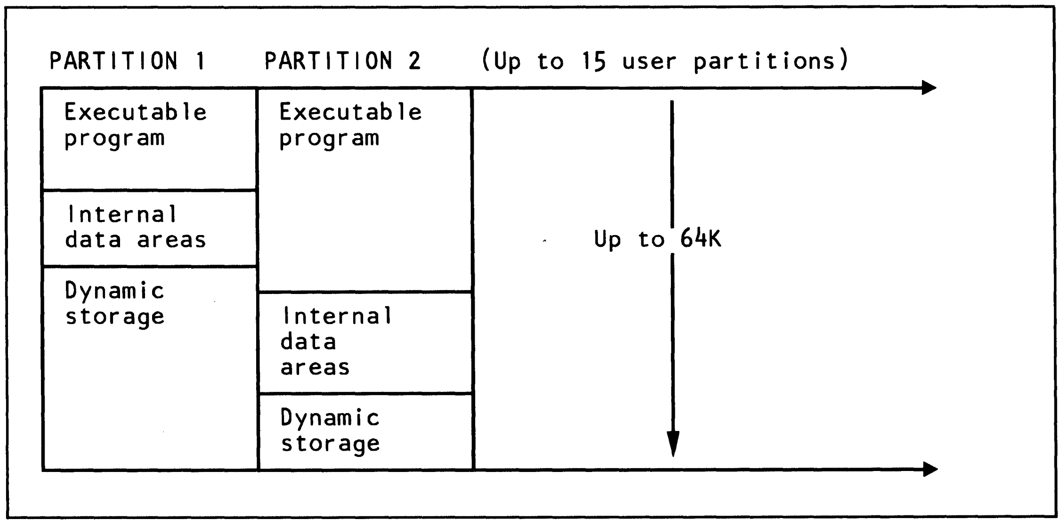


Figure 2-1. Example of Storage Organization

The operating system allocates a quantity of processor storage (logical storage) to a partition. A **partition** is a portion of logical storage dedicated to the execution of one task set at a time. A program executes within a partition of storage. Programs executing in one partition do not interfere with programs executing in another partition. At each node, the

Realtime Programming System Supervisor executes in partition 0, which is reserved for it. This partition uses two address spaces: one for its data and one for its instructions.

You define the quantity of processor storage (logical storage) that is to be allocated to each partition. You can define up to 15 partitions at each node for your programs.

Partitions provide storage space for an executable program and for internal data areas. If the partition size you define exceeds the storage required to hold your program and its internal data, the excess storage in the partition is used as dynamic storage.

Dynamic Storage: Dynamic storage is the available logical storage left in a partition after a task set is loaded. Dynamic storage services allow your program to access dynamic storage during execution. Your program can use dynamic storage for data areas or buffers. For example, the Series/1 PL/I compiler is designed to run in a 26K-byte partition. The storage required for the compiler program itself is 20K bytes. The additional partition storage is used for dynamic work areas. The compiler can use storage in excess of 26K bytes to speed up execution by allocating larger program work spaces. Using dynamic storage allows you to write a program that can adapt itself to use additional storage.

Defining Primary Storage

Primary storage is defined for the system data address space (SDS), system instruction address space (SIS), static partitions, and dynamic partitions.

System Data Address Space: The **system data address space** (SDS) is the part of the system task set that contains the system data areas for the Realtime Programming System. This part of the system task set resides in address space 0. No user task sets can reside in address space 0.

System Instruction Address Space: The **system instruction address space** (SIS) is the part of the system task set that contains the system instructions for the Realtime Programming System. This part of the system task set resides in address space 1. No user task sets can reside in address space 1.

Static Partitions: **Static partitions** are defined when the system is installed or by executing an operator command, DEFP, during IPL. A static partition is fixed in size and remains defined until the system is reloaded. The operating system executes in a static partition.

Dynamic Partitions: **Dynamic partitions** are defined when needed. The operating system creates a dynamic partition for you:

- When a program (task set) is being started and no partition number is specified, or
- If a partition number is specified that has not been defined as a static partition

The operating system determines the size of your program (rounded up to the nearest 2K bytes), allocates a partition of that size, and then loads and executes the program. If insufficient storage is available to create the partition, your program is not executed until enough storage is available for the partition. The operating system frees a dynamic partition's storage when the program completes execution.

Your program is placed into execution faster in a static partition because the partition's storage is allocated at IPL time. Static partitions are more efficient but require planning on your part. You should use static partitions when you plan to run production programs of a predictable size that do not change, especially those that must execute on a regular basis.

Dynamic partitions are more flexible, but using them requires slightly more time to place your program into execution. For example, a program that executes in a dynamic partition might not be able to execute immediately because the system is unable to allocate enough physical storage. This situation could occur if several dynamic partitions are currently active.

Managing Contention for a Partition

All programs are queued (lined up) for execution. A program is queued for execution in the requested partition, and the program waits until the partition is free. By queuing programs, the operating system manages contention for partitions when multiple programs (task sets) attempt to use the same partition.

Queuing Priority

The operating system selects which program to start based on the priority of the programs in the queue (their **queuing priority**). Programs of the same priority are selected in a first-in-first-out manner if resources are available. You select a queuing priority for a program in order to control the order in which programs execute in a partition. The queuing priority for programs is unrelated to the dispatching priority for tasks.

Extending Partition Storage

You can extend partition storage by using disk overlays, loadable external modules, transients, secondary storage services, and additional storage services.

Disk Overlays

Disk overlays are overlay segments that reside on direct access storage devices and, when called, are loaded into the overlay area of logical storage associated with their resident segments. Disk overlays are described under "Overlays" on page 5-30.

Loadable External Modules

Loadable external modules are a type of program segment that is loaded into dynamic (logical) storage or a storage pool from a direct access storage device. Loadable external modules are built by the application builder and resolved to their owning task sets so that they can be executed. See the section entitled "Loadable External Modules" on page 5-34 for more information.

Transients

A **transient** is a program or data module that is loaded from disk into processor storage. Transients allow task sets to operate in a smaller amount of storage than would otherwise be needed. Transients are of two kinds: system and user.

When a system transient is required, the operating system loads the transient into processor storage from disk or diskette. After the transient completes execution, its storage or transient area is available to another transient. By using dynamic transient pool management (DTPM), the operating system retains frequently used system transients in storage. This process allows the operating system to optimize performance on the basis of the storage you provide in the pool and on actual system transient usage. The system programmer decides the amount of storage to assign to the dynamic transient pool when the system is installed.

User transients are self-relocating programs that run in problem state in a user partition. These programs are not part of the resident portion of the task set, but are loaded by the LOAD macro from disk into dynamic storage in the user partition.

For more information on system and user transients, see the *Supervisor Services Programming Guide*.

Secondary Storage Services

The supervisor provides secondary storage services which you can use to extend storage capacity. **Secondary storage** is processor storage outside a defined partition. Secondary storage is called unmapped storage because it is not mapped in an address space by segmentation registers.

You can allocate secondary storage by requesting that an amount of unmapped storage be assigned to a partition. On request, portions of secondary storage are mapped and made available to a program. Secondary storage is mapped by setting a logical address (segmentation register) to create an area of selected blocks of physical storage.

Secondary storage services include:

- Storage overlays
- Storage records
- Task storage records
- Control module mapping
- Dynamic transient pools
- SNA buffer pools

By using these services, which utilize secondary storage, you avoid the more time-consuming process of reloading from disk or diskette.

Storage Overlays: A storage overlay is an overlay segment that resides on a direct access storage device and is loaded into secondary storage at task set load time. The largest storage overlay associated with each composite module is included in a storage overlay area within the task set load module, which is read into primary storage at task set load time. When called, a storage overlay is accessed through manipulation of the segmentation registers assigned to the overlay area associated with its resident segment. Storage overlays are discussed under "Overlays" on page 5-30.

Storage Records: Your programs can also allocate and map secondary storage for **storage records**. You specify an amount of secondary storage to be associated with a partition or task set for storage records. When a storage record is defined, a number of 2K-byte blocks of unmapped storage are assigned to it, and it is mapped to your partition. When the record is mapped, you can store data in it.

The storage record remains addressable to your task set until you request addressability to a new storage record, or until you request a record that you have previously created. Because the data that you write in a storage record is retained until the task set terminates, you can gain fast access to this data by invoking the supervisor to map an existing storage record.

Task Storage Records: Storage records are associated with task sets. Within a supervisor-state task set, however, you can associate a storage record with an individual task. A storage record associated with a task is called a **task storage record**. The system maps the task storage record into the same area of logical storage each time it dispatches the associated task.

You can define several task storage records, each associated with a different task in the task set. The system maps the task storage records, one at a time, into the same area of logical storage each time it dispatches the associated task.

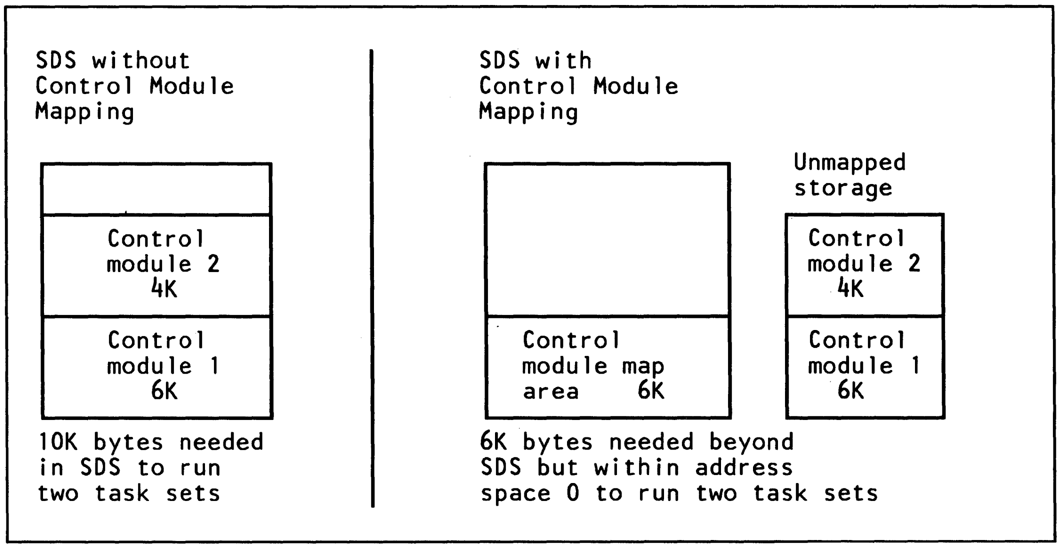
When you purge a task storage record, the system no longer maps the task storage record when it dispatches the associated task.

Control Module Mapping: You can logically extend supervisor data address space (SDS) beyond its 64K-byte limit by using a facility called **control module mapping** for non-supervisor control modules. To use this facility, you allocate a control module mapping area beyond SDS, but within address space 0. The amount of logical storage you should allocate for the mapping area must be equal to or greater than the largest control module of the task sets you plan to execute. If you use a shared task set, the mapping area must be large enough to contain both the sharing and shared task set's control modules simultaneously.

Non-supervisor control modules are located in unmapped storage. The supervisor sets the control module mapping area to map the control module for the task set that it selects for execution. Mapping is accomplished by setting the processor segmentation registers to the secondary storage address of the control module. No data movement occurs.

Control module mapping extends the capacity of the system because all user control modules of active task sets need not be simultaneously mapped in a single 64K-byte data space. Logical storage is required only for the largest control module. Storage in SDS that is no longer being used to hold control modules is available for control blocks needed to support additional devices or communications lines. Thus, large configurations can be supported within the 64K-byte addressing limit.

The savings in SDS is illustrated below.


SDS without Control Module Mapping
Control module 2 4K
Control module 1 6K
10K bytes needed in SDS to run two task sets
SDS with Control Module Mapping
Unmapped storage
Control module map area 6K
Control module 2 4K
Control module 1 6K
6K bytes needed beyond SDS but within address space 0 to run two task sets

Dynamic Transient Pools: To improve system performance, you can request that secondary storage be allocated to logical storage pools that the operating system uses to hold system transients. These areas are called **dynamic transient pools**. A dynamic transient pool can only be allocated to the operating system partition.

SNA Buffer Pools: SNA buffer pools are data areas required by Realtime Programming System SNA support. They are system-managed and reside in secondary storage.

Additional Storage Services

Additional storage service facilities are available for the following licensed programs or facilities:

- Indexed Access Method
- Multiple Terminal Manager
- Storage pools

These storage service facilities are discussed under the topics "Indexed Access Method" on page 5-35, "Managing the Application with the Multiple Terminal Manager" on page 5-25, and in the *Supervisor Services Programming Guide* for storage pools. Note that Indexed Access Method index pages can be either in secondary storage or on disk, as can Multiple Terminal Manager user transaction programs, and user-controlled storage for storage pools. Figure 2-2 on page 2-13 shows the various ways logical storage can be extended.

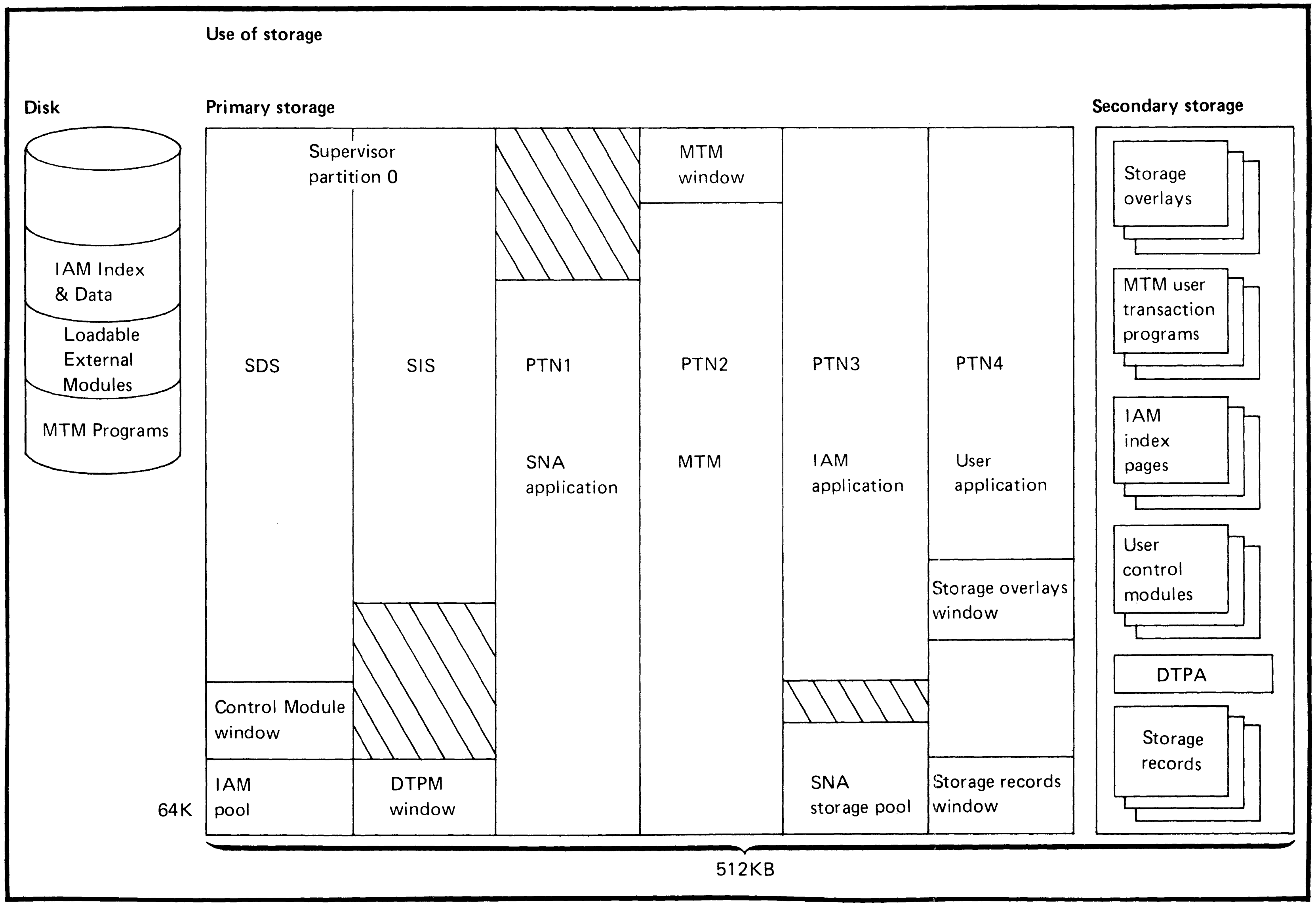


Figure 2-2. Ways to Extend Logical Storage

Managing Data and Devices

An important aspect of any data processing application is the entry, updating, and retrieval of data. System input and output (I/O) is accomplished through the data management and device management facilities provided by the operating system. Refer to the *Data Management Programming Guide* for complete information on system I/O support.

Device Management

Device management services within the operating system support various devices for input and output. In particular, the operating system supports the following I/O devices:

- Non-removable disks
- Removable diskettes
- 9-track magnetic tape devices
- Line and matrix printers
- Display terminals and teletypewriters
- Binary synchronous communications
- Start/stop (asynchronous) communications
- Series/1 support for the IBM Systems Network Architecture (SNA)
- Sensor I/O devices
- Timers

For disk and diskette, each physical record (sector) has a relative block address that is the physical address of the sector on the device.

For non-storage devices (such as a printer), you must structure your data by specifying the number of bytes (characters) to transfer. The number of bytes must fit the physical record size of the device.

Terminals

You communicate directly with an application or with the operating system using a terminal. The operating system and various licensed programs support the following types of terminals:

- IBM 3101 Display Terminal
- IBM 4978 Display Station
- IBM 4979 Display Station
- IBM 4980 Display Station

You can also develop a system that supports an additional terminal, or a terminal emulator program (that is, a virtual terminal), as the operator console or as a display device. Existing display applications may be able to use the additional terminal (or terminal emulator program) as their I/O device with no change.

Local/Remote Transparency

An important facility when you use a 3101 Display Terminal is local/remote transparency. Your 3101 can be attached directly to one of the nodes in the system, or it can be attached remotely over communication lines. In either case, your program is made independent of the line protocols that are needed when communications lines are involved. You can use a remote 3101 as an operator console or as a display terminal for the command language facility.

Online Device Tests

An operator, using commands, can test active devices. The online device tests allow an operator to test disks, diskettes, printers, floating-point hardware, timers, and most terminals. You use the data created by the online tests to help decide when IBM must service the system.

Data Management

Data management services support input and output (I/O) to files. A file is the fundamental I/O structure that the operating system uses. The term applies to data written to or read from any I/O device.

The collection of data that is written to a device is termed the output file. The collection of data read from a device is called the input file. An input or output file for a non-storage I/O device is simply the collection of data read from or written to the device between the opening and closing of the file.

Symbolic Names

In general, the first concern in accessing data is referring to the device for I/O. For ease of use, devices are referenced by **symbolic names** (for example, PRINTER, DISK, DISPLAY, etc.) instead of by a number representing the absolute node and device addresses. Using this technique, a program need not be sensitive to the details of the hardware installation or be concerned with the details of configuration management.

Decisions about the relationship between actual devices, their physical addresses, and the symbolic device name can be made:

- Dynamically using a system operator command
- Using a system macro (STARTDEV)
- When the SYSGEN program is executed
- With the configurator program

The standard system supplies a set of default device assignments. These defaults are discussed in the *Standard System Installation Guide.*

If your system has support for at least one device of a given device type (for example, a 4963 disk), you can readily add support for another device of that same type (4963 disk). The operator enters an operator command (REDY) to ready the device for use or a start device operator command (STDV) for the new device. The operating system creates the needed control data for the new device from the existing device and starts it. This facility is called dynamic device generation. Dynamic device generation allows you to expand the configuration of a node without extensive preplanning. You also avoid the need to run the

SYSGEN program to add the new device to the system. The device is added dynamically and is available for application programs to use immediately.

The next concern in accessing data is referencing a file. A file is referred to by symbolic name. The operating system uses the symbolic name when the data comprising the file is read from or written to the device. For example, when a program reads input from a disk file, the operating system uses the symbolic name to search for and locate the data on the disk. You do not need to know the actual device-dependent location of the file because the operating system allocates the space for the file when the file is created and maintains its location for future accesses.

Associating a File with a Program

The operating system provides services that allow a program to access files. The program must establish a connection between itself and the file. The operating system's OPEN service does this.

Data Set Definition (DSD)

The program must also identify how it will use the file. This is done by defining the data structure using a **data set definition** (DSD). The data set definition allows you to specify a variety of file attributes such as:

- File name
- Record format (how records are to appear within a block)
- Size of a logical record
- Size of a block
- Device name (where the file resides)
- Direction of access or access mode (input only, output only, or both input and output)
- Access level — the level of support you need from the system I/O routines:
  - To block data on output, deblock data on input, and manage data buffers in processor storage (GET/PUT level)
  - To transfer blocks of data to and from disk or diskette with no automatic buffering, blocking, or deblocking (READ/WRITE level)
- Access method (whether data is transferred sequentially or directly)
- File organization (consecutive, random, or partitioned)

A data set definition can be made part of an assembler language source program. In this case, changes can be made in data formats or file names either during program execution or by reassembling the program.

Data Set Definition Table (DSDT)

The operating system also allows you to define your data set definitions (DSDs) outside of your program. These DSDs are placed into a **data set definition table** (DSDT), a file that is associated with your program at execution time. DSDTs exist at several levels: task set, shared task set, or local level (meaning at the system-resident volume of a node), or global level of a multiprocessor system. This approach to defining DSDs provides greater flexibility because you can change a DSD in your DSD table without changing your program. You can use the command language facility, job stream processor, or system utility commands to create or modify the necessary DSD table entries. In this case, the program makes a symbolic reference to this data set definition when it issues the OPEN for a file. The OPEN processing picks up the necessary file attributes. This approach is sometimes called late association because the file information is determined at execution time. In contrast, with early association, the file information is incorporated into the source program at compile or assembly time.

Programs that you create in PL/I, Fortran, Pascal, or Cobol use the external data set definition technique and late association to give you greater flexibility in writing application programs. Once a file is opened, it remains open until the task closes it explicitly or completes execution. When a task completes execution (either normally or abnormally), the operating system dissociates any files that were open when the task completed.

Where Data is Stored

A file consists of data written to or read from an I/O device. Operating system services provide a variety of file structures and flexible data storing techniques.

Data can be stored on disk devices or system formatted diskettes. When you initialize a disk or diskette, you prepare the disk or diskette in system format.

You can refer to the data on these devices by using the:

- Device
- Volume
- Data set
- Member

Device

A **device** refers to an entire disk or diskette device. A disk or system-formatted diskette[2] has room for at least one volume.

Volume

A **volume** is a group of related data sets on a disk or diskette. Each device and volume has a **table of contents** area that identifies all files that are stored on the device or volume.

[2] A diskette can be in system format, basic exchange format, type E exchange format, or type H exchange format. A disk is always in system format.

Data Set

A **data set** is usually the file structure to which input and output is performed. A data set must reside in a logical volume. A data set that is structured as a set of related files is called a **partitioned data set**. Each file in the partitioned data set is called a **member**. A partitioned data set has a table of contents similar to the table of contents found in a logical volume.

For files created on disk or diskette, you define the size of a file before using the file for output. The operating system reserves the space you request and records the file name, file organization, and file boundaries in the table of contents of the device, volume, or partitioned data set. When the file is used, the space allocation does not increase dynamically if you attempt to write more data to the file than it can hold. If a file is not filled on output, you have the option of releasing the unused space.

How Data is Structured for Storing

Logical Records

Typically, you want to read and write records of data in your application program without being concerned about how the data is stored on disk or diskette. The operating system allows you to read and write **logical records**. To do this, you use the GET/PUT level of access discussed in the section entitled "Access Levels" on page 2-22. At the logical record (GET/PUT) level, the operating system structures your logical records into blocks for output and deblocks the data into logical records on input.

You specify the format of the logical records, the maximum size of each record, and the maximum size of each block in the file. You control the size of a block and the number of logical records grouped into each block. Choosing optimum block sizes maximizes I/O performance and gives you flexibility in using disk space. The way you select record formats, sizes, and blocking schemes affects the efficiency of the use of disk or diskette storage.

Record Format

The format of each logical record is either **fixed** in length or **variable** in length. Fixed means that the same number of characters appears in each logical record. Variable means that the number of characters in each logical record can vary.

Record Size

The size of a logical record is the number of characters in the record. For records in variable format, the operating system uses four bytes at the front of each record to indicate the record's length. When you specify a variable record size, you include the length of the data (number of characters) plus the four bytes of length information. Files containing variable length records can not be accessed directly, because the location on the disk of a record or block can not be determined from the relative block address.

Block Size

Data is written to or read from disk or diskette in **blocks**. The size of a block can be larger or smaller than a sector, but the optimum block size is a multiple of the sector size. For blocks containing variable format records, the operating system uses four bytes at the front of each block to indicate the block's length. When you specify a variable block size, you include the length of the longest record plus the four bytes of block length information.

In general, you achieve efficiency in space use if the block size is a multiple of the sector size.[3]

Spanned Records

When using fixed or variable format records, you maximize space use on disk or diskette by spanning records across blocks. This technique wastes no space because entire blocks are used to store data. You must group spanned records into blocks. An example of how data is structured for storing is shown in Figure 2-3 on page 2-20 .

[3] For disk, the size of a physical record (sector) is 256 bytes. For diskette, the size of a physical record (sector) can be 128 bytes, 256 bytes, 512 bytes, or 1024 bytes.

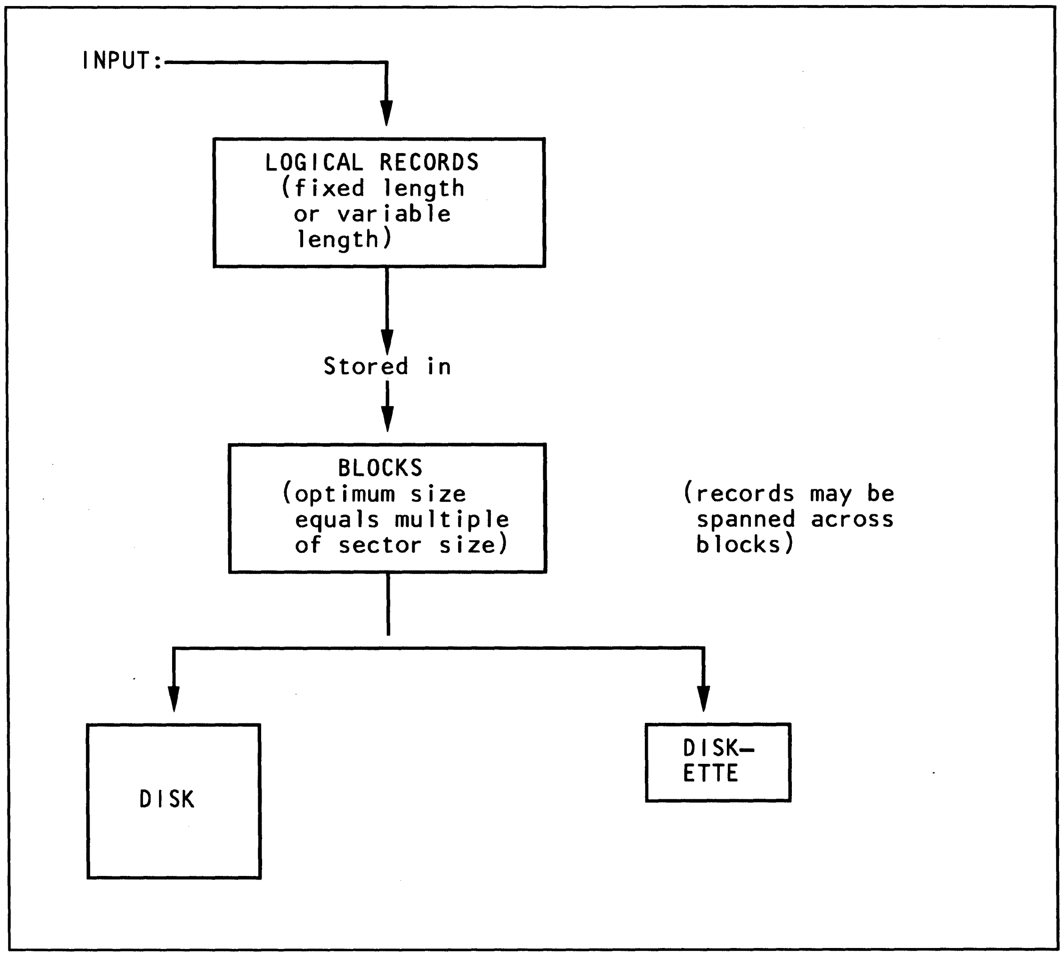

INPUT:
LOGICAL RECORDS (fixed length or variable length)
Stored in
BLOCKS (optimum size equals multiple of sector size)
(records may be spanned across blocks)
DISK
DISK-ETTE

Figure 2-3. Example of How Data is Structured for Storing

How Data is Organized Within a File

The operating system supports several file organizations. An example of how data is organized within a file is shown in Figure 2-4 on page 2-21 .

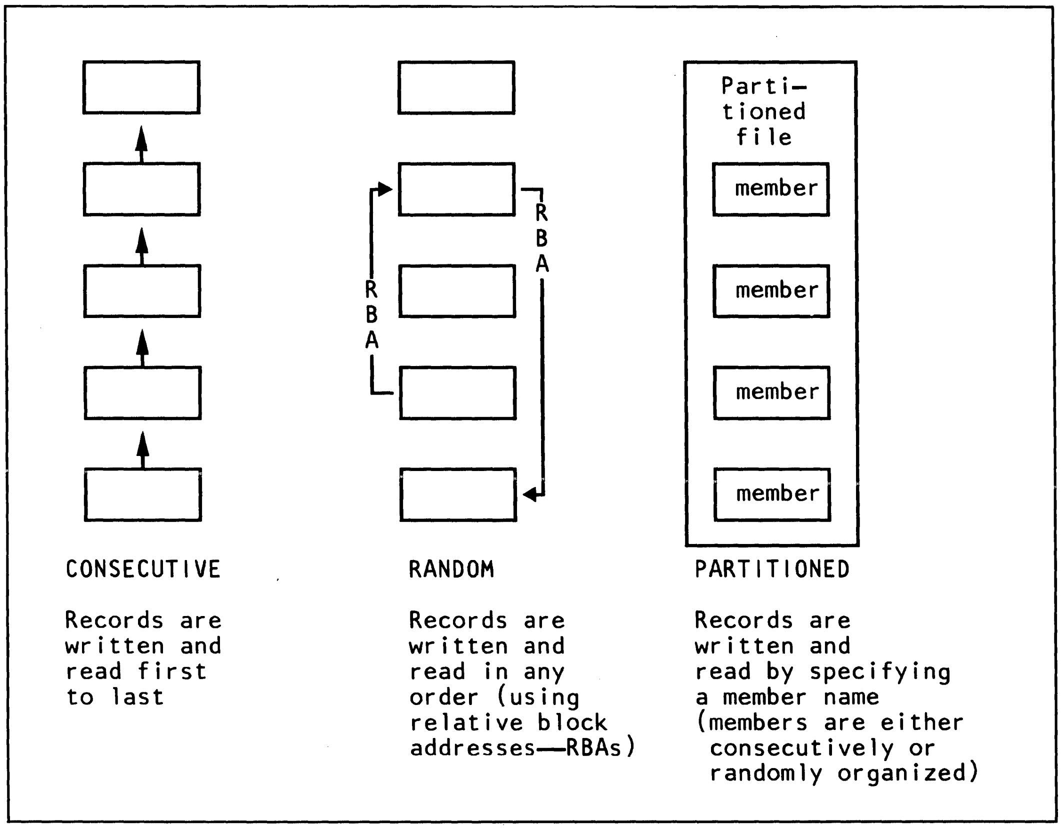

Parti-
tioned
file
member
member
member
member
RBA
RBA
CONSECUTIVE
Records are written and read first to last
RANDOM
Records are written and read in any order (using relative block addresses—RBAs)
PARTITIONED
Records are written and read by specifying a member name (members are either consecutively or randomly organized)

Figure 2-4. Examples of Data Organization Within a File

Consecutive Organization

All devices permit files that are organized **consecutively** — where blocks are written one after the other. The records are retrieved in the same order in which they were written (that is, first to last).

Random Organization

Disk and system formatted diskette devices also permit files to be organized **randomly** — where blocks are written in any order. For random files, physical addresses called relative block addresses are assigned to each sector on the disk or diskette. Your program reads or writes a block of data in any order in the file by specifying the relative block address of the desired record.

Partitioned Organization

Disk and system formatted diskette devices also permit partitioned data sets. A file organized as a **partitioned** data set is a file that contains a set of related files called members. You read or write data to a partitioned data set by specifying a member name for a file to be accessed within the data set. Members within a partitioned data set can have either consecutive or random organization.

Indexed Organization

An **indexed** file is a special random file in which records are stored and retrieved by using a portion of each record called a key. You assign the location and the length of the key. The records are later retrieved by specifying the symbolic key. The Indexed Access Method licensed program provides support for indexed files.

Access Levels

The operating system supports three levels of input and output called **access levels**. Each level does different things for you.

Logical Record Level (GET/PUT)

Typically, you want to read and write records of data in your application program without being concerned about how the data is stored on disk or diskette. At the logical record level, you read and write logical records. The operating system structures your logical records into blocks of one or more records for output and deblocks the data into logical records on input. It also creates processor storage buffers for input and output.

The logical record level is also called the GET/PUT level. GET and PUT are the system service (macro) names for reading and writing logical records.

You specify the format of the logical records, the maximum size of each record, and the maximum size of each block in the file. Because the size of a block is under your control, you control the number of logical records that are grouped into each block. This control lets you choose optimum block sizes that maximize I/O performance and that give you flexibility in using disk space. The flexibility in choosing record formats, sizes, and blocking schemes uses space on disk or diskette with varying degrees of efficiency.

Physical Level (READ/WRITE)

The next lower level, the physical level, is the programmer's interface to most features of IBM-supported I/O devices. At the physical level, you must be aware of the characteristics of the I/O device.

The physical block level is also called the READ/WRITE level. READ and WRITE refer to the system service (macro) names for reading and writing physical blocks.

Basic Level (EXIO)

The lowest level, the basic level, gives you access to any hardware function of any I/O device supported by the operating system. To use this level of I/O, you must write your program in Series/1 assembler language.

The basic level is also called the EXIO level. EXIO is the system service (macro) name for reading and writing data at the basic level.

Device Independence

The greatest degree of device independence is achieved:

- By not specifying device characteristics that are unique to a particular device, and
- By using the highest level of I/O (the logical record level)

In most cases, the PL/I, Pascal, Fortran, and Cobol compilers determine the proper access level based on your description of the file in the data set definition.

How Data is Transferred (Access Methods)

Access methods (or access types) are techniques for moving data between processor storage and I/O devices. The access methods for reading or writing data are:

- Sequential
- Direct
- Indexed (Indexed Access Method only)
- Keyed direct (Multiple Terminal Manager only)

Sequential and direct access are part of the standard operating system support. Indexed access requires that the Indexed Access Method program be installed. The Multiple Terminal Manager facility supports keyed direct access.

Sequential Access

With **sequential access**, blocks are read or written in sequential order. On input, data is accessed from the first block until the end-of-data indicator is detected. On output, blocks of data are written from the first block until the ending boundary of the file is detected.

Direct Access

Records or blocks accessed **directly** can be read or written in any order within the physical boundaries of the data set. Records or blocks must be in fixed format.

Indexed Access

The Indexed Access Method accesses a record using a key that must match some part of the actual record. Indexed access can be used if a file is being accessed by key or sequentially.

Keyed Direct Access

Keyed direct files have an alphameric key in positions 0 to n of each record. Keyed direct is a fast means of accessing records by keys and is useful if you do not intend to access an indexed file sequentially. Keyed direct files are special random files that are only supported under the Multiple Terminal Manager.

Direct Access for Display Terminals

In addition to supporting direct access on disks and diskettes, the operating system provides direct access mode for display terminals. Your program can use line character positions as values to read or write portions of the display. The values specified for line and character positions must be within the bounds of the display area allocated to the data set.

Ensuring the Integrity of Data

To ensure data integrity, the operating system allows volumes to be "duplexed." A **duplex volume** is a logical volume of which the system keeps two copies. Duplex volumes may reside only on disk and the two copies must be kept on separate physical devices. In contrast, a **simplex volume** is a volume on disk that is not duplexed.

Duplexing volumes is an optional service that improves data availability and integrity. If the devices that contain the two copies of a duplexed volume are on the same node, duplexing provides backup data in case of disk failure. If the devices are on separate nodes, duplexing provides backup data in case of disk or node failure. In general, you duplex volumes that contain critical data. For example, volumes containing key system data sets (such as the global data set definition table) should be duplexed. Duplexing is not available for system residence volumes.

Duplexing is done at the logical volume level; not at the physical (device), data set, or member level. Individual data sets or members of a partitioned data set are duplexed if they are part of a volume that is duplexed.

In order to use duplexing, you must relate your devices to one another with the following naming convention: the names of the two devices are the same except that one is prefixed by a dollar sign. For example, the system expects a disk device named $DISKA to be defined in order to back up duplex volumes on the disk named DISKA. This naming scheme allows Version 6 to remain compatible with previous versions of the Realtime Programming System.

When creating a volume you would like to duplex, you can create the volume with the duplex option. You can also change a simplex volume to a duplex volume, or vice versa, using the command language facility command named CONVERT. For more information on the CONVERT command see the *Command Language Facility User's Guide*.

Recovering Down-Level Duplex Volumes

If a WRITE operation to a duplex volume fails to update both copies of the data, the system records this event in the error log and directs all subsequent I/O activity on the affected volume to the remaining good copy. The other copy is considered down-level.

The command language facility command named DVCHECK can be used to determine if both copies of a duplex volume are identical.

You can restore the two copies to the same level using the command language facility command named RECOVER. RECOVER can run while other programs are using the current level copy; CONVERT cannot. For more information on the RECOVER command see the *Command Language Facility User's Guide.*

Ensuring System Integrity and Availability

When multiple applications are executing (multiprogramming or multiprocessing) the failure of a single program must not affect the successful execution of other programs. The following operating system facilities ensure integrity among executing programs and ensure availability of the Series/1.

System Address Space

The operating system runs in a separate address space, which is protected from user address spaces. This protection prevents any program from inadvertently overwriting supervisor programs and data.

Control Blocks

The operating system data areas needed to execute and manage programs are called **control blocks**. Control blocks are collected in a separate data area called a **control module**. When a program (task set) is loaded, the control module for each program is made accessible to the supervisor. The control modules are protected from inadvertent damage because this storage is not accessible to other user programs. Protection of your control module allows the operating system to recover system resources from abnormally terminated programs. This protection ensures that other programs will continue executing or that other programs can be loaded for execution.

System-Wide Function and Execution

In a multiprocessor system, you can execute a function in nodes throughout your system. For example, duplex volumes at different nodes are useful if one disk fails. Similarly, if a node fails, you can access the devices attached to that node from another node, using the manual Two-Channel Switch.

A set of common terminals can be bi-directionally switched, under program control, between two nodes. This dynamic reconfiguration can recover these common terminals from a failing node, and return them when appropriate. This Programmable Two-Channel Switch can also be manually operated.

The Two-Channel Switch and the Programmable Two-Channel Switch are mutually exclusive; that is, they cannot both be installed on the same pair of nodes.

Name Constants

System control blocks are indirectly addressed by application programs using system-generated names rather than storage addresses. A system-generated name is a **name constant** or NCON. This technique prevents your program from damaging the supervisor if an invalid control block address is specified.

Releasing Resources

The operating system provides a set of routines that are invoked when any secondary or primary task terminates. These routines are designed to release resources that are allocated to the task or task set so that the resources can be used by subsequent programs. For example, when a task ends, any files that were opened by that task are closed.

Task Error and Termination Exits

The operating system provides a task error exit and a task termination exit facility that allows programs to implement special termination clean-up processing. PL/I, Cobol, Fortran, and Pascal support these clean up facilities. Thus, you can ensure that your application program terminates in an orderly fashion even when it fails.

Address Space for User Partitions

User partition addressability is restricted to a single address space (except in the special case of a shared task set). Thus, applications that execute in separate address spaces are protected from one another.

Problem/Supervisor State Execution

User programs are executed in problem state. Executing in **problem state** prevents a program from executing privileged instructions that could destroy other user programs or data in the system partition.

In contrast, programs that execute in **supervisor** state can execute any privileged or non-privileged instruction. Supervisor-state programs execute only within the supervisor partition.

System-Handled I/O

The operating system performs all input/output operations. The boundaries of files and the buffer areas used in all transmissions are examined to ensure that I/O operations are performed in a fashion that ensures correctness and avoids inadvertent access to or destruction of data and storage. Full I/O error recovery facilities are supported to diagnose and retry failing I/O operations. If I/O operations fail, the operating system performs I/O error logging to record the relevant data about the failure.

Subscript Range Checking

Programs developed using PL/I, Fortran, and Cobol can be compiled to contain subscript range checking. This facility prevents you from inadvertently destroying your executable program area or data area.

Program Development

Standard System

An easy-to-use program development environment is important to ensure successful definition, creation, and support of production environments. The standard system that is shipped from IBM is pre-built at IBM to provide you with an operating environment that is easy to install and begin the definition and creation of your applications. The standard system with the Program Preparation Subsystem is optimized for systems dedicated to program development.

High-Level Languages

High-level languages — PL/I , Fortran, Pascal, and Cobol — are available. Extensions for each language give you access to many operating system functions. A macro assembler language with structured programming macros and a library of macros that interface to most system functions is also available.

Program Preparation Subsystem

The Program Preparation Subsystem provides most of the necessary tools for developing, testing, and maintaining applications. Two of these tools are the text editor, a full-screen editor with commands that invoke a variety of editing functions for entering and maintaining source language files or data files; and the application builder, a facility for preparing programs for execution under the Realtime Programming System. The function of the application builder is similar to that of linkage editors of other systems. Two command languages are available — an interactive command language supported by the **command language facility** and a batch processing **job control language** supported by the job stream processor.

Command Language Facility

The command language facility provides commands that access most system functions, create and maintain files, and compile and prepare programs written in a high-level language or in assembler language.

Utility Packages

The operating system and many licensed programs provide utility packages that allow interactive use of their unique set of functions at a terminal. In addition to the interactive stand-alone, system, and tape utility packages, the following licensed programs provide interactive utility functions:

- Indexed Access Method
- Multiple Terminal Manager
- Query

Program development tools and utility functions are discussed in greater detail in Chapter 5, "Development Tools and Application Aids."

Sharing Data, Programs, and Devices

An application is usually divided into a set of cooperating but separate tasks or programs (task sets). To support this environment, the operating system provides a set of functions that allow controlled sharing or exclusive use of programs, data, and devices.

These resources can be shared locally within a task set or between a shared task set and the task set(s) that are using the shared task set. Resources can also be shared globally in both uniprocessor or multiprocessor systems and synchronized by using global resources.

You control the use of program modules in a task set and the sharing of program modules within that task set by specifying that a program module is:

- Reentrant
- Serially reusable
- Nonreusable

You specify the above attributes for the primary program of the primary task in a task set.

The PL/I, Pascal, and Cobol compilers and the macro assembler allow you to produce reentrant programs. The Fortran compiler does not produce reentrant programs.

Reentrant Program Modules

A program module is **reentrant** if it can have concurrent executions active without requiring each usage to terminate its execution before another can start. Reentrant programs have the following attributes:

- Instructions in the program are not modified during execution
- Separate, variable, data areas are provided for each invocation of the program

The program module and task management components of the operating system support reentrant programs. If several requests are made to execute a subroutine as a task from within a program (task set), the operating system recognizes this condition and permits execution. An example of this is several RUN statements for the same procedure in a PL/I program.

Serially Reusable Program Modules

A program module that is executing as a **serially reusable program** has only one active invocation of the program executing at a time. A serially reusable program can be shared, but the currently active execution of the program must complete before another user starts the program. A serially reusable program can modify its data, but the program must restore its initial state before another user uses it.

Nonreusable Program Modules

A **nonreusable program module** is a program that modifies its instructions or data areas during execution and does not restore its initial state before completing. Nonreusable programs cannot be executed more than one time. If the program must be executed again, you must use a fresh copy. Nonreusable programs are appropriate only when you know that the program will execute infrequently and is not shared by other programs. A nonreusable program is always loaded from disk or diskette each time that it executes.

Sharing of Resources Across Partitions and Nodes

Often, resources must be shared among several programs (task sets) executing in different partitions or among cooperating tasks executing in the same partition. You can use shared task sets and your ability to link task sets to accomplish this as well as global queues and global resources to share data among several task sets. Distributed devices, global queues, and global resources are called system-wide resources. System-wide resources are shared by an entire uniprocessor or multiprocessor system.

Shared Task Sets

You can share programs and data across partitions in a single node by defining one or more special task sets called **shared task sets**. A shared task set is a specially built program that you designate as containing data and subprograms that are to be accessible to active programs (task sets) in other partitions in the same node. When a program of this kind is built, a copy of its external symbols, the **task set reference table** (TSRT), is saved. A shared task set executes in a static partition.

Once you create a shared task set, you can write application programs that refer to data and subprograms that reside in the shared task set. You prepare your program for execution and designate that any unresolved external symbols are to be resolved to the designated shared task set. This symbol resolution process is performed by the application builder. In addition to resolving your program's external references to the shared task set, the application builder also marks your programs as needing the specified shared task set for successful execution. This process tells the operating system to make sure that the needed shared task set is active when a task set requiring a shared task set is loaded for execution.

Subprograms that you place in a shared task set should be reentrant if you wish to support full sharing. Examples of full sharing are the PL/I and Cobol transient subroutine libraries. These transient libraries execute as shared task sets and are used by all PL/I and Cobol applications.

Subprograms that are written in PL/I, Pascal, Fortran, Cobol, or assembler language can be placed in a shared task set. Any program that references the shared task set can use the subprograms. Your program can use resources (such as an event definition) of the shared task set.

You can also share data across several programs that execute in the same node by including it in a shared task set. The data could reside in a global storage area or a system queue. If the data is read-only, then you need provide no control over access to it. If the data is not read-only, you must serialize access to the data. Serialization is done using the REQUEST/RELEASE functions of the operating system. PL/I provides access to these system functions using the LOCK/UNLOCK statements. For Fortran, these functions are provided in the Fortran Realtime Subroutine Library.

Details on using a shared task in PL/I, Pascal, Fortran, or Cobol applications are given in the user's guide for each language.

Linking Task Sets

An application consisting of several related task sets may require that two task sets execute one after another without any intervening tasks sets executing. To accomplish this, the system provides the facility for one task set to invoke (be linked to) another task set. The operating system terminates the execution of the linking task set. The next task set that is executed is the linked task set. The two task sets communicate using a global area within the partition.

Global Areas

Global areas are allocated in a special area of your partition. Global areas are accessible from any task within the partition.

A global area is not overlaid during a special circumstance when a task set is linked to another task set to be executed next in the same partition. The global area of the partition is not changed when the linked task set is loaded for execution. Instead, references in the linked task set to the global area are resolved to the data items contained in the global area definition in the linked task set. The actual data accessed by the references in the linked task set is the data placed in the global area by the previous task set (the linking task set).

PL/I programmers use global areas by specifying uninitialized STATIC EXTERNAL data and by specifying the TRANSFER TO statement. Fortran programmers use GLOBAL data to use global areas and link to another task set using the INVOKE function. Assembler language programmers use the GLOBL data type and the LINKTS system service to define and use global areas.

Local Queues

You use local queues to share data within a task set or among task sets (if the queue is defined by a shared task set) executing in the same node. A local queue can only be added to by the task set that defined it or by task sets that share the task set that defined it. A local queue may be private or public depending on whether or not tasks other than the defining task (but within the defining task set) can take elements from it.

Global Queues

You can share data among several programs in different partitions and at different nodes (in a multiprocessor system) using **global queues**. You can take advantage of this form of sharing without using a shared task set. The global queue is defined by the program that will take data elements from the queue. Only this program can take elements from the global queue. However, any program can put elements on the queue. Global queues can reside in processor storage or on disk. Additional information is provided in the section entitled "Inter-Task Communication" on page 2-35.

Controlled Sharing of Files and Devices

Controlled Sharing of Files

Data within files stored on a disk or system formatted diskette can be accessed by any program requesting it unless the file was opened with exclusive use requested. When exclusive use is requested, the operating system allows only one application to access the file and prohibits access by all others.

Controlled Sharing of Interactive Terminals

The operating system provides a unique function: controlled sharing of interactive terminals by multiple applications. Operating system I/O routines for display terminals (the 4978, 4979, and 4980 terminals) enable you to associate a program function key with an application. Furthermore, you can designate part of a display screen or the entire display screen as a file from which input is read or to which output is written. The file appears as a window on the screen for application I/O. A **window** is one or more adjacent lines that use part of a display screen or the entire screen.

An end user uses multiple applications at a display screen by pressing the program function key associated with the appropriate application and then entering input lines within the window defined for the application. You ensure that windows do not overlap by defining window files with unique line numbers. You prevent multiple applications from sharing a display terminal by opening the device for exclusive use.

Controlled Use of Indexed Files

The Indexed Access Method program controls access to records in indexed files via system-wide record locking. Record locking means that all requests to update a record in an indexed file are locked out until the application in control of the record completes its update.

Writing Realtime Applications

Process Control

Developing a realtime application is challenging because its success usually depends upon providing fast response and designing strategies that must handle a wide variety of situations in short periods of time. One of the most demanding kinds of realtime programs is the process control application. The term **process control** usually means that the program is tied to a process or procedure that is executing in production mode and whose control must be performed in realtime. That is, they are cyclical in that the time periods used are "wall clock" time, not time in execution.

As an example, a Series/1 system might be used to control the flow of parts in a manufacturing process. Sensors on a conveyor belt might enable the system to recognize and count parts and to direct them to the manufacturing area in which they are needed. In such an application, the system must respond to the presence of a part and sort it before it proceeds beyond a certain point on the conveyor belt. The whole process is then geared to the speed of the conveyor belt and how long it takes to recognize a part and direct it to its needed area. Usually, a process control application involves reading and writing external sensors; responding to process interruptions from sensors; handling the expiration and initiation of service cycles, based on external definitions or time of day; and in coordinating and controlling a number of separate activities.

The following sections discuss system facilities that help you in producing a realtime application. Many of these are general services that are applicable to other kinds of applications.

Synchronizing Tasks

Often, the execution of a given task cannot proceed until other tasks have performed certain operations. In these situations, the task that cannot proceed needs to have its execution synchronized with other cooperating tasks.

Task synchronization can be achieved using:

- Events
- Serially reusable resources

- Timer services

Events

Task synchronization is usually achieved using **events**. A task uses the operating system WAIT function to suspend itself until a given event occurs. Using the operating system POST function, another task signals the occurrence of the event to the waiting task. A task may wait for one or more events to occur before proceeding with execution. Waiting for a list or set of events is common in most realtime applications.

The operating system recognizes several different kinds of events:

- Events associated with the completion of input or output operations, including response to operator input
- Events associated with the occurrence of process interruptions
- User-defined events that you associate with activities you define

Events in the first two categories are set complete (posted) by the operating system when the associated action completes. User-defined events are set complete by explicit user action using the operating system POST function. Tasks that synchronize with user-defined events must execute on the same node of a multiprocessor system.

Serially Reusable Resources

You can also accomplish task synchronization using the operating system serially reusable resources (via the REQUEST/RELEASE facilities). Serially reusable resources are used when you deal with situations where several tasks need to use a piece of data, a file, or a section of code that can only be used by a single task at a time. In this environment, each task agrees to associate a specific resource name with access to the facility in question. Each task then requests control of the resource before it attempts to access it. The operating system only grants the resource to a single task at a time. Other tasks requesting the resource can choose to be placed in the wait state until the resource becomes available or can request notification if the resource is busy. In the latter case, the requesting task does other work and requests the resource again later. It is important to emphasize that this form of synchronization is only enforced by user programming conventions.

The system provides both node-wide and system-wide serially reusable resources. Node-wide serially reusable resources are used when serialization on a per node basis is sufficient. System-wide or global serially reusable resources are used when the serialization must be done for the entire multiprocessor system.

Timer Services

Tasks can also be synchronized by having a task delay its execution for a specified time interval or until a particular time of day occurs. This process is done using timer services.

Scheduling Tasks and Task Sets for Execution

You schedule tasks and task sets to let the operating system know when you want to execute them. In a multiprocessor system, tasks and task sets may be scheduled in any node. The operating system provides two types of scheduling: immediate or delayed.

Immediate Scheduling

For immediate scheduling, the operating system is instructed to execute the task or task set as soon as it is possible to do so. The immediate scheduling of tasks is done using the system function STARTASK.

Delayed Scheduling

For delayed scheduling, the execution of the task sets is deferred until a particular event occurs. The types of events supported for delayed scheduling are:

- Schedule a task set on the occurrence of a process interrupt
- Schedule a task set to execute after a time interval
- Schedule a task set to execute at a particular time of day
- After a certain time of day, execute a task set at a user-specified interval

The operating system directly supports delayed scheduling of task sets. The system scheduler function uses a special data area known as the system scheduler table to contain entries for the task sets that are to be scheduled. Delayed task sets can also be unscheduled using a system function to remove entries from the system scheduler table.

Delayed scheduling can be simulated by starting the task and then causing the task to suspend itself until the appropriate event occurs. Series/1 PL/I supports this approach. PL/I also supports the scheduling of tasks on the occurrence of process interruptions, at a particular time of day, or after a time interval has expired.

Stopping the Execution of Tasks and Task Sets

The system provides services to stop the execution of tasks and task sets. These services are important because stopping a task or task set may be the only way to recover its resources for other programs to use. When a task or task set is stopped, it has the opportunity to enter a designated error exit to perform any necessary special actions. The termination processing of a task or task set in the operating system is designed so that resources are returned to the operating system. In particular, termination has the following effects:

- Resources held by the task or task set are released
- Files that are opened are closed
- Time intervals or events that are defined in the task are cancelled
- Control blocks used to monitor the task's execution are freed
- Partition storage for a task set using a dynamic partition is freed

Inter-Task Communication

An important point to consider when implementing a multitasking or multiprocessing application is how data can be passed between asynchronously executing tasks. Either shared files or queues, or both, allow data to be passed (exchanged) between tasks.

Shared Files

In using shared files, the file to be shared is opened in one task and is accessed by several other tasks. Simultaneous access can be used successfully if the following conventions are followed:

- The file should be accessed directly using either relative record numbers or the Indexed Access Method
- If you are using the Indexed Access Method, it ensures that updated records are stored in the file so that only the current copy of a record is maintained. If you are not using the Indexed Access Method and you are writing your application in assembler language, use the READ/WRITE level of access. You cannot use the GET/PUT level of access. In PL/I, use SHARED files.
- If several tasks are to update records in your application, use the Indexed Access Method, if possible, because it provides automatic record locking to protect you from simultaneous updates. An alternative is to use operating system services (WAIT/POST or REQUEST/RELEASE) to provide your own record level locking.

Queues

A second way to communicate data between tasks is to use the operating system **queuing** services. These services allow several tasks to send and receive data using a symbolic name. The data you send must not exceed 4094 characters. The area where the queue resides can be totally within processor storage, totally on disk, or a combination of storage and disk.

You can define a **queue** as local or global. A local queue can only be accessed by tasks in the same partition or by tasks in other partitions of the same node if the queue is allocated by a shared task set. If sharing among task sets is required, you should use the global queue function. A global queue can be accessed by any task set that knows the symbolic queue name.

You can further define a local queue to be private (where only the defining task can take elements from the queue) or public (where any task in the defining task set can take elements from the queue).

A useful feature of operating system queuing service is that system queue functions allow a receiving task to be suspended until a message arrives at a designated queue. This facility allows you to easily design and implement an application so that the application can be activated when work arrives for it to process.

Disk queues also have a restart attribute that allows the operating system to restart them if necessary. The operating system restarts the processing of the queue with no elements (a cold restart) or with all elements from previous processing intact (a warm restart).

Realtime Applications Using Sensor I/O

Many realtime applications must exchange data with non-computer sources. An example of such a requirement might be the reading of an electronic temperature gauge by a program.

The operating system allows you to read and write both digital and analog devices. The programming interface to sensor I/O devices allows you to refer to the device and the points with symbolic names. PL/I supports sensor I/O with language extensions to record I/O. Fortran supports sensor I/O via callable subroutines in the Realtime Subroutine Library.

Finding and Repairing Program Defects

Before an application program is put into productive use, its defects must be found and repaired. The process of finding program errors, repairing them, and retesting is known as debugging. This section outlines those facilities that the operating system provides and the supporting programs that aid you in the debugging process.

Finding Problems Before Execution

The most productive type of debugging is often that which is accomplished before a trial program execution. This type of debugging is facilitated by the extensive diagnostic facilities that are provided by the language processors which are offered under the operating system. Each language processor contains extensive source diagnostic facilities that are aimed at eliminating syntax and semantic errors. Use of these facilities to eliminate compile time errors ensures that you can move to the trial execution step in a productive fashion.

Finding Problems During Execution

Various facilities help you find incomplete logic or faulty design in your application coding. First, there are the facilities that are available to the various high-level languages. Second, there are facilities that are available to all users. These facilities are of particular benefit to those programmers who are writing assembler language programs.

High-level Language Debugging Aids

PL/I, Fortran, Pascal, and Cobol provide execution time diagnostic aids that help you pinpoint a source program failure and correct it. Of particular importance are the following facilities:

- Object time error exits
- Execution tracing
- Subscript range checking
- Execution time messages

Object Time Error Exits

The Cobol and Fortran languages allow you to write object time error exits. These error exits are defined by error handling clauses or statements for many conditions such as I/O error conditions. You should employ these facilities so that you can easily diagnose exceptional conditions in your program.

The PL/I language supports the concept of error exits in a much more general way. You write ON units in your application program to handle the vast majority of errors you could encounter. The ON unit facility allows you to provide specific messages or program logic to recover from error situations.

Execution Tracing

The language implementations on a Series/1 provide different forms of execution path flow tracing. This type of facility is a valuable way of determining the execution path of an application. Flow tracing is implemented by Cobol, Fortran, and the S1/EXEC interpreter. PL/I provides a calling execution trace via the SNAP option of the ON statement.

Subscript Range Checking

Often in PL/I or Fortran applications, execution time loops are set up where subscript values of arrays are computed and used to store values. If these subscript values are out of range (exceed the established bounds of the array), using them can destroy other user data or code in the application. You prevent this occurrence by using the Fortran debug statements or by enabling the PL/I subscript range condition. If either of these actions is taken, the compiler generates additional code to validate subscript values before any damaging usage occurs.

Execution Time Messages

PL/I, Cobol, and Fortran are designed to provide you with execution-time error messages when your application encounters an error condition and your program does not contain error-handling source coding. These messages help you spot problems without resorting to machine level problem determination. The PL/I compiler is especially helpful in this area. PL/I's execution time diagnostic support permits you to pinpoint the failing source statement.

Machine Level Problem Determination

If your application is written in Series/1 assembler language, or if you are familiar with Series/1 assembler language and you are using either PL/I or Cobol, a variety of problem determination aids are available to you. These aids include:

- Generated machine code listings
- Dumps

- Online debugging
- Traces
- Online terminal test
- Online test
- System error log
- Displays
- Time stamping of messages
- Full-screen editor (FSEDIT) to examine and change data in hexadecimal format

Complete information on problem determination is in the *Problem Determination* manual.

Generated Machine Code Listings

Both the PL/I and Cobol compilers produce annotated assembler language listings of the generated code for your program. You can employ these listings in conjunction with other facilities mentioned in the section to perform problem determination, when other source level debugging aids have proved ineffective.

Dumps

The principal value of a dump is that it allows you to inspect both the data and instruction areas of your partition and to determine the state of various program variables. A dump is useful in doing detailed analysis away from the machine.

The operating system provides three facilities for dumping the contents of storage — the DUMP and SNAP services and the processor storage-to-diskette dump utility. The DUMP and SNAP services dump (display) the contents of your partition. The processor storage-to-diskette dump utility is a stand-alone utility that dumps the contents of processor storage to diskette.

In most cases, the dump you get using the DUMP service is produced on abnormal termination of a task. The dump is written to a member of a file named ABDUMP. The ABDUMP member contains a complete dump of a partition as well as system control blocks that your application was using.

Using the SNAP service, you can dump selected portions of a partition without terminating your program. SNAP writes its output directly to a printer.

The processor storage-to-diskette dump utility enables an operator to dump the contents of processor storage on diskette when the diskette containing the utility is loaded. Loading the dump program destroys the contents of storage addresses 0 through 255, so record the contents of these bytes manually before using this utility.

In a communications environment, you can transmit a dump from a remote Series/1 to another Series/1.

Online Debugging

The operating system offers an interactive multiuser debug package that assists you in locating errors in an assembler language program. DEBUG aids in testing multitasked programs in a multiprogramming and multiuser environment. By operating a program under control of DEBUG, you can:

- Stop the program each time execution reaches any of the instruction addresses which you specify. These addresses are known as **breakpoints**. You can have multiple active breakpoints in a program.
- Trace the flow of execution of instructions within the program by specifying that DEBUG enter into the single-step mode. Then, each time the program executes an instruction, the terminal may optionally display the contents of the instruction address register (IAR), the address key register (AKR), the level status register (LSR), general registers, and the op-code before the execution of each instruction.
- Continually monitor the contents of a specific storage location, and enter DEBUG when that location is modified
- Restart program execution at the breakpoint, or restart program execution at other than the next instruction
- Each time program execution reaches one of your breakpoints, you can:
  - Display registers and storage location contents
  - Print control blocks
  - Modify the contents of storage locations and registers
  - Display or modify the contents of storage media
  - Print storage location contents

By using these functions, you can determine the results of computations performed by the program and the sequence of instructions executed within the program. You can also modify data or instructions of the program during execution.

The debug package can also be used with programs produced with high-level languages. However, effective use requires knowledge of Series/1 assembler language and the implementation techniques of the language processors.

Traces

The operating system supports several types of traces:

- Supervisor call (SVC) trace
- Communications trace
- I/O trace

The supervisor call (SVC) trace is a trace of all SVC instructions issued in a particular partition. An SVC trace is useful because the system services that your application uses are invoked using the SVC instruction. You use the SVC trace instruction to determine the SVCs that your program executed successfully as well as the last successfully executed SVC.

The communications trace is a trace of all asynchronous (start/stop) and binary synchronous I/O over a set of communications lines. This trace is extremely useful for tracing communications line problems in applications using data communications.

The I/O trace is a trace of I/O requests to the devices that the operating system controls. This data is useful in locating I/O errors that are causing an application to fail.

Online Terminal Test

You can use online terminal testing to verify proper operation of terminals and communications lines. Online terminal testing aids in diagnosing line or terminal trouble.

Online Test

The command language facility command, OLT, enables you to perform online tests of the devices and features available on your system. Use this command to determine if a device is operational; not to diagnose specific failures. The OLT command starts the device or feature by exercising a basic set of functions.

System Error Log

The system error log data set is used to record error conditions that the operating system detects. The conditions recorded include IPL, node failures, node start ups, changes in the status of duplex volumes, errors in data transfer between nodes, program checks, machine checks, and I/O errors.

Using the system utilities, you can produce a formatted report of the information in the error log data set. You can optionally specify what types of errors are to be reported from the log data. The log data aids in problem determination. It will help you to analyze why a particular program or piece of hardware is failing.

The IBM customer service representative who services your Series/1 also uses the error log information to assist in locating and repairing a failing hardware component.

Displays

The operating system provides an operator command called DISP that allows a system operator to request that the system display information about system activity and status. You use this command to determine:

- What partitions are active and what programs they contain
- What write-to-console instructions have outstanding replies
- What storage is available
- What programs (task sets) are queued for execution
- What programs (task sets) are in the system scheduler table
- What devices are available
- What is the time of day and date
- What are the spooled jobs and spool writers
- What programs have disk locations known to the system scheduler
- What are the internode communication facility trace table entries
- What are the internode communication facility statistics
- What is a node's status
- What are the disk seek statistics
- What is the status of subsystems or terminals known to the terminal controller

Time Stamping of Messages

All system messages contain the date and the time of day.

Full-Screen Editor (FSEDIT)

FSEDIT is a full-screen editor for altering data in hexadecimal format. Blocks of data (up to 256 bytes) on disk or diskette are examined and changed, according to your instructions. You can save a block of data, cancel editing a block, print either a block or just the changes you have made, and find and/or replace a string (character or hexadecimal).

Customizing Your Operating System

You may want to customize the operating system to support only functions you require while using a minimum amount of processor storage. Customization of the operating system is usually necessary if a system that is different from the standard system is needed. A customized system can be generated for a uniprocessor system or for one or more nodes of a multiprocessor system configuration. You customize a system by:

- Executing the configurator program
- Changing the IPL options file or User Input Command (UIC) file used at IPL time
- Dynamically generating devices
- Executing the SYSGEN program provided with the Program Preparation Subsystem
- Using the INSTALL option of the IPLMAINT utility to change the size of the dynamic transient pool area, the system data space and instruction space, or the size of the control module mapping area

The Configurator

The configurator is the program you use to set up or change the make up of a system. You start the program using a command language facility command. (The command language facility is an interactive part of the operating system consisting of menus and commands for programming and managing files.) Once you enter the command, menus appear on which you type the needed information to configure your system.

The configurator enables you to:

- Display the configuration of nodes
- Give each node a name (or accept the default name), and designate whether or not the node should be brought online automatically when you perform an IPL of the system
- For each node:
  - Designate the task set to be used as the system task set at the node
  - Allocate storage to the system and to partitions
  - Display the devices at the node, give them names (or accept the default names), and examine and set device-dependent options for each device
  - Designate whether or not the Indexed Access Method is to run at that node, and if so, specify parameters for it

  - Assign devices to the terminal controller
  - Establish characteristics for Systems Network Architecture (SNA) support at the node
  - Display and modify other system parameters
- Copy a custom-built system to the node

IPL Options File and User Input Command (UIC) File

The IPL options file and the User Input Command (UIC) files are special files that contain operator commands that are executed when the operating system is loaded and started. You can change characteristics of the operating system by editing these files, entering the appropriate operator commands, and then restarting the system. This approach is an easy way of customizing the standard system to meet your needs.

Dynamic Device Generation

Once a system is installed, you can dynamically change the number of most devices that were generated in the system without regenerating and reinstalling the system. This enables you to start another device of any type that is already defined in the operating system that you wish to modify. Once the support exists for a type of device, any other device of the same type can be started.

SYSGEN Program

The SYSGEN program is supplied with the Program Preparation Subsystem. SYSGEN is a question and answer program that builds a specification file that is used to customize your operating system. The specification file contains all information that is necessary to generate a complete system.

The specification file that was used to generate the standard system is shipped with the SYSGEN program. Because SYSGEN provides an update mode that allows you to change the IBM-supplied specification file, you may only need to update a few of the specifications to meet your particular system requirements.

Complete information on customizing the operating system is located in the *System Customization Guide*.

System Operation

The operating system provides facilities that allow an operator to monitor the status of the system, to ensure productive use of the Series/1 processors, and to install IBM-supplied system patches. Operator commands can be issued from the operator console, a remote terminal, or be included in your program. Command language facility commands also allow system users to use operating system facilities that ensure productive use of the processor (such as printer output spooling).

Operator Facilities

An operator, through the use of a full set of operator commands, can control the execution of programs, query the status of all programs in the system including the control program, allocate resources and control the use of the resources, use system debugging and problem determination facilities, and control system routines that write spooled output to a printer.

Typically, an operator loads the operating system initially by pressing the LOAD button on the hardware control panel.[4] On a multiprocessor system, the operator needs to press LOAD only on the control panel for the home node. The system will start the remaining nodes. The operator then enters commands to set the time of day and date and to define the command language facility as a subsystem to the terminal controller.

Complete information on controlling system operation is located in the *Operation Guide and Reference* manual.

User Control Facilities

Operating system spooling functions enable you to write printer output at high speed to disk. Command language facility commands support printer output spooling. The spooled output support enables you to request multiple copies of printed output or to request that output be printed on different forms.

The command language facility also allows you to queue task sets for execution or to display the status of submitted jobs.

Information on controlling system operation using command language facility commands is located in the *Command Language Facility User's Guide*.

Installing IBM-Supplied Patches

The patch application tool is an interactive, menu-driven application that allows you to install IBM-supplied patches to the operating system or to other licensed programs. It also provides a module replacement capability. Using the patch application tool you can:

- Apply corrections from an IBM-supplied diskette
- Apply IBM-supplied patches entered at a display station or teletypewriter device
- Remove patches that were previously applied by the patch application tool
- Print the amount of system patch area used and available
- Print patch log data sets
- Print the patch data for a specific authorized program analysis report (APAR)
- View and modify the SYSVOL table
- Prepare the system task set for IPL
- Delete the old system task set

Information on using the patch application tool is located in the *Problem Determination* manual.

[4] The IPL command can be used to perform an IPL of a Realtime Programming System from an executing Realtime Programming System. Binary synchronous communications enable you to perform a remote IPL of a Realtime Programming System.

Reporting Problems with IBM Licensed Programs

If you have a problem with a Series/1 program, see the *Software Service Guide* for instructions on how to report the problem and obtain resolution.

Chapter 3. Data Communications and Network Support

Modern computer systems often consist of several processors of varying types that are interconnected using communications lines. Such systems are said to be decentralized because all the system's data processing functions are not handled by a single, central computer but are distributed among several processors. Significant physical distances often separate each interconnected processor. Also, the devices (terminals, printers, etc.) that communicate directly with each processor may be distant from the processor. This decentralization of computing resources in a system allows data processing functions to be distributed among multiple processors while the results of the processing are known throughout the system.

Contents of this chapter:

Communications Applications

Applications that use communications can make use of the multitasking facilities of the operating system to separate the communications task from the processing tasks of the application. This separation of processing and communications allows you to choose the proper programming language (high level or assembler) for specific processing tasks. The processing task reads and writes messages via subroutine calls to the communications task and inter-task communications facilities.

The communications task is usually an assembler language program containing assembler language macros that define and control communications lines and devices. The macros allow you to:

- Define the characteristics of remote devices and associated communications lines
- Transmit and receive messages
- Break connections
- Establish a list of remote device identifiers for communications with dial-up (switched) lines

An application communicating with a remote device sets up the message sequence using an acceptable communications protocol and then passes the message to the hardware interface using cycle steal I/O commands.

A communications system can be dedicated exclusively to one application, or the communications system can be used as a system I/O device. In the latter case, you use remote devices or processors as you would any other I/O device connected directly to the Series/1 processor.

The Realtime Programming System data communications components and support for Systems Network Architecture (SNA) environments allow a variety of communications applications including:

- Decentralized processing
- Protocol conversion
- Message concentration
- Data communications in a network
- Front-end processor

Decentralized Processing

Data processing environments for a business organization are typically large in number and are often remote from one another and from the center of the business enterprise. Distributed systems evolved to match this decentralized data processing environment.

The Realtime Programming System facilities for remote communications support decentralized processing environments that:

- Require batch store and forward processing. For example, a Series/1 could be used to collect transactions entered at an interactive terminal, store the data on disk, and then connect regularly to a host system to transmit the data to a central data base. You can develop batch store and forward applications or use facilities provided with the Communications Manager for the Series/1.

- Perform remote transaction processing that allows inquiry and update of a central data base on a host system. A data base, of which the host system keeps an image, might also be used at the remote site. Remote transaction processing uses a Series/1 processor connected to display terminals and optional line or matrix printer to emulate an IBM 3270 Information Display System. The Multiple Terminal Manager program supports remote transaction processing through the Multiple Terminal Manager 3270 device emulation. The Communications Manager for the Series/1 also supports remote transaction processing.

- Permit you to write programmable logical units to communicate messages in a Systems Network Architecture (SNA) network. Such a network might communicate with the Information Management System (IMS) or Customer Information Control System (CICS) executing on a host computer. If you do not develop your own programmable logical units, the Communications Manager for the Series/1 supports this kind of message exchange for you.

Protocol Conversion

Realtime Programming System facilities and supporting licensed programs enable a Series/1 processor to communicate with a variety of non-Series/1 devices that use differing protocols. The 4987 Programmable Communications Subsystem Preparation Facility and Extended Execution Support enable you to develop applications that communicate with the variety of devices that often exist in a communications environment. The Communications Manager for the Series/1 also provides protocol conversion support.

Message Concentration

Message concentration applications collect transactions entered at interactive terminals and prepare messages in the proper message format for transmission to an application executing on a host system. The Communications Manager for the Series/1 supports message concentration and takes care of routing messages to their final destination under operator control.

Data Communications in a Network

The Realtime Programming System enables a Series/1 processor to communicate with other processors and devices in a Systems Network Architecture (SNA) network or connected to an X.25 packet switching network.

The X.25/HDLC Communications Support enables a Series/1 to connect through a packet switching network to other Series/1 processors, or to act as a node within an X.25 network.

The operating system support and extended support for SNA networks enables you to write your own applications that communicate with host systems. A communications environment using the Systems Network Architecture permits the transfer of data based on the demand transactions entered at an interactive terminal.

The Synchronous Communications Single Line Control / High Speed Adapter enables a Series/1 to connect to an X.21 leased network or an X.21 circuit switched public data network. Data transmission uses synchronous data link control communications (SDLC) or binary synchronous communications (BSC) in half-duplex mode.

Front-End Processor

The need to enter batch jobs remotely for execution on a host computer and the need for remote data entry at remote terminals resulted in the use of a Series/1 system as a front-end processor for the host system. Such a system uses one or more front-end processors to collect data and batch jobs being submitted from remote devices and to prepare the information for processing by the host computer.

Communications Support

The architecture of the Series/1 enables you to physically connect a variety of remote devices (including processors) to Series/1s over communications lines.

The data communications support is logically connected to operating system input/output support and includes the operating system's error detection and recovery services. For example, the operating system supports the online testing of local and remote terminals to ensure that messages are transmitted properly.

The operating system opens files that establish communications with a terminal or processor. Your communications application then transmits data using the operating system READ/WRITE level of access.

Operating system routines and the communications interface hardware generate and control message transmission for a particular communications link.

The Series/1 hardware interfaces supporting communications use:

- A cycle-steal channel to access message sequences in processor storage for transmission and to insert message sequences into main storage. Use of cycle stealing results in minimum processor interaction and overhead.
- Microprocessors to diagnose errors. The microprocessors control the communications interfaces and ensure self-diagnosis of communications line problems.

Communications Connections

A **communications connection**[1] physically connects devices in a communications environment. Communications environments transmit information electronically, usually through telephone lines in a public telephone system.

A **point-to-point** communications connection physically connects a Series/1 to a remote device. Devices in a point-to-point connection use leased (nonswitched) telephone lines or dial-up (switched) telephone lines through modems. For dial-up lines, the connection is maintained only for the duration of the communication. If a point-to-point connection uses a dial-up line, one control station (such as a Series/1) can communicate with several remote stations after a link has been established between the control station and each remote station.

A **multipoint** communications connection physically connects a control station with multiple remote stations (called secondary stations). All stations in the multipoint connection are physically connected over a single leased line.

1 The terms communications connection, data link, communications link, and communications line are used synonymously in this book.

Communications Protocols

A communications protocol is the form or convention by which particular sequences of characters are interpreted and acknowledged over a communications line.

The Realtime Programming System supports the following communications protocols:

- A variety of asynchronous (start/stop) communications protocols
- Binary synchronous communications
- Synchronous data link control (SDLC) communications
- X.21 interface
- X.25 interface

Asynchronous (Start/Stop) Communications

Data communications facilities in the base operating system support asynchronous communications. A variety of device protocols are supported in half-duplex mode.

Series/1 PL/I directly supports asynchronous communications using record I/O.

Binary Synchronous Communications

Data communications facilities in the base operating system support binary synchronous communications. The binary synchronous communications protocol is used most often in exchanging data and programs between computers, between remote job entry terminals and computers, and similar applications. Binary synchronous communications enable you to:

- Transmit either EBCDIC, ASCII, or transparent EBCDIC data
- Perform block checking for error detection
- Remotely IPL Realtime Programming Systems

The binary synchronous communications protocol operates in half-duplex mode using point-to-point or multipoint connections.

Series/1 PL/I directly supports binary synchronous communication using record I/O.

Synchronous Data Link Control (SDLC) Communications

Systems Network Architecture (SNA) support in the Realtime Programming System enables communications using the synchronous data link control (SDLC) protocol. SDLC protocol is the standard link-level protocol for communicating in an SNA network. SDLC operates in half-duplex mode using point-to-point or multipoint connections at line speed rates up to 9,600 bits per second.

X.21 Network Support

The SDLC communications protocol supports non-switched half-duplex connections of a Series/1 to an X.21 network through an SNA interface. Operations on X.21 circuit switched public data networks are supported in binary synchronous and SNA/SDLC modes in compliance with the CCITT X.21 recommendation.

X.25 Interface

The X.25 protocol is supported by means of the X.25/HDLC Communications Support licensed program. The X.25 support is discussed in the section entitled "X.25/HDLC Communications Support" on page 3-14.

Systems Network Architecture Support

The IBM Systems Network Architecture (SNA) is the communication architecture subscribed to by many IBM programs and communication products. SNA support in the Realtime Programming System is an implementation of a subset of the total IBM Systems Network Architecture. The support provides system services to establish, control, and terminate sessions between multiple Series/1 programs and System/370 subsystems or user programs.

The support also provides system services to transfer data and control information between the programs. The SNA network consists of the following elements:

- A Series/1 system defined as a cluster controller with multiple logical units and associated user applications
- A System/370 consisting of a virtual operating system using Virtual Telecommunications Access Method (VTAM) or Telecommunications Access Method (TCAM) and user applications
- An IBM 3705 Communications Controller with its Network Control Program connecting a Series/1 to the host processor
- Synchronous data link control (SDLC) communications line between the Series/1 and the 3705 controller

Systems Network Architecture Extended Support

The Systems Network Architecture Extended Support consists of a set of assembler language macros that enable you to write applications that communicate with the Information Management System (IMS/VS) or the Customer Information Control System (CICS/VS).

The support extends the basic SNA support and masks SNA protocols from an application program.

Additional Communications Support

In addition to the base operating system support for communications, the following licensed programs provide additional support for communications:

- 4987 Programmable Communications Subsystem
- Advanced Remote Job Entry
- SNA Remote Management Utility Programming RPQ
- Series/1 to System/370 Channel Attach Program
- 3270 device emulation using the Multiple Terminal Manager
- X.25/HDLC Communications Support
- Remote Manager
- Communications Manager for the Series/1

4987 Programmable Communications Subsystem

The 4987 Programmable Communications Subsystem is a separate processing facility that supports applications involving large numbers of terminals and communications lines. Often the device requirements in communications applications and the characteristics of the devices vary considerably. The Programmable Communications Subsystem supports a variety of communications lines, line speeds, terminal types, and communications protocols (including non-IBM devices and protocols). The subsystem is adapted to the Series/1 hardware and software architecture so that the same type of self-diagnosis, availability, and error recovery features as in the operating system are supported.

The Programmable Communications Subsystem allows you to perform the following types of communication processing on the Programmable Communications Subsystem hardware instead of on a Series/1 processor:

- Redundancy checking
- Generation and recognition of control characters
- Data chaining to and from processor storage
- Telephone call answering and originating
- Timeouts or interval timer
- Modem control
- Console control
- Self-diagnosis and tracing functions
- Automatic polling

If the above tasks were handled on a Series/1 processor, significant overhead could occur when large numbers of different communications lines and terminals are used.

The Programmable Communications Subsystem provides the following interfaces:

- Synchronous and asynchronous EIA interfaces
- Automatic call handling interface
- Teletype[2] current loop interface
- Synchronous and asynchronous integrated modems

[2] Trademark of the Teletype Corporation

The Programmable Communications Subsystem allows you to have a variety of line speed and interface types and to mix them arbitrarily within the communications subsystem.

The Programmable Communications Subsystem uses a scanner to scan each interface to collect or transmit characters. The scanner also provides:

- Programmable synchronization and line turnaround characters
- Programmable selection of bits per character
- Parity checking
- Programmable selection of the number of stop bits for asynchronous terminals

The controller within the Programmable Communications Subsystem allows you to customize the communications subsystem to handle a particular group of terminals and a particular application. The controller provides:

- A protocol level instruction set
- The capability to customize the communications subsystem to line type, protocol, and functional level
- Parameter build capability for management of the communications subsystem

You use controller functions by using an assembler language macro library that supports two types of macro instructions:

- Communication macro instructions used to customize communications programs (called function strings) for each line of the Programmable Communications Subsystem
- Communication definition macros used to define tables, parameters, line types, function strings, and pointers that the subsystem needs

Preparation Facility

The 4987 Programmable Communications Subsystem Preparation Facility is used to support the generation of controller storage image programs for the IBM 4987 Programmable Communications Subsystem. The program provides a macro library for assemblies with the Program Preparation Subystem.

Extended Execution Support

The 4987 Programmable Communications Subsystem Extended Execution Support is used to write application programs that communicate with devices that are attached via the 4987 Programmable Communications Subsystem. Two levels of user interface are supported; READ/WRITE and EXIO.

Documentation

- *4987 Programmable Communications Subsystem Preparation Facility Reference*
- *4987 Programmable Communications Subsystem Extended Execution Support Reference*

Advanced Remote Job Entry

The Advanced Remote Job Entry program enables you to use the Realtime Programming System as a remote job entry work station. The program can be used in an SNA network or in a network using multileaving binary synchronous communications.

Using binary synchronous communications, the Series/1 system can communicate with a System/370 that uses any of the following host entry subsystems:

- OS/VS2 JES2
- OS/VS2 JES3
- VM/370 RSCS

Using SNA, the Series/1 can communicate with a System/370 that uses any of the following host entry subsystems:

- OS/VS2 JES2
- OS/VS2 JES3
- DOS/VSE VSE/POWER

The Advanced Remote Job Entry program enables you to transmit one or more batch files to a System/370 job entry subsystem for processing. The files are System/370 job streams. Upon completion, the output from the job stream(s) normally is routed to a Realtime Programming System data set or to a Series/1 printer.

Multileaving Remote Job Entry

The binary synchronous option supports point-to-point (switched or non-switched) communication using the multileaving remote job entry technique. **Multileaving** enables input and output data streams to be intermixed on the communication line. The data transmission is fully synchronized and two-directional. A variable number of data streams can be transmitted between the Series/1 system and the host system. The Series/1 system appears as a System/3 with console support to the host job entry subsystems.

SNA Remote Job Entry

The SNA remote job entry option supports point-to-point or multipoint (switched or non-switched) Synchronous Data Link Control lines. The option includes support for up to four logical unit sessions at a single work station. The option uses Logical Unit Type 1 protocols for session communication with the host job entry subsystems.

Unattended Operation

The following Advanced Remote Job Entry facilities allow a work station to operate unattended:

- Commands in a disk or diskette data set
- Dynamic punch file allocation
- Delayed session activation

An Advanced Remote Job Entry user can place remote job entry commands in a disk or diskette data set or enter them at the work station display terminal. Commands in a data set are read in and processed just as if they were entered at the display terminal.

You can also activate the Advanced Remote Job Entry program in a wait state. Delayed activation means the connection is established with the host job entry subsystem only when a call is received from the host.

Punched output received from the host is always placed in a Realtime Programming System disk or diskette data set. The Advanced Remote Job Entry program allocates this data set dynamically.

Other Work Station Functions

The work station console function lets you query the host system for the status of a submitted job. You can also query the host for any other normally allowed information (i.e., system status etc.). Other facilities allow you to record console activity in a Realtime Programming System data set for subsequent printing. This is called journaling.

Printed output sent from the host is either printed directly on a Series/1 printer or, through spooling, is placed in a disk or diskette spool data set for printing at a later time. You use Realtime Programming System spool facilities to control the printing of specific jobs from the spool processor output queue.

Advanced Remote Job Entry supports 3211/3203-4 Forms Control Buffers (FCB) for printed data. You use a utility to define FCBs corresponding to the FCB requests that may be sent by the host.

Commands

Advanced Remote Job Entry commands are single line commands with parameters. These commands are summarized below:

- Attend - changes operational mode (attended/unattended)
- Help - prints a list of the commands
- Journal - turns journal activity on or off
- Library - changes library environment
- Operator - transmits a host operator command
- Print - changes the current printer information
- Punch - changes punch information
- Readfile - identifies a command or data file to be processed
- Status - reports current work station status

Documentation

Complete information on the Advanced Remote Job Entry program as used with the Realtime Programming System is located in the *Advanced Remote Job Entry User's Guide.*

SNA Remote Management Utility Programming RPQ

The SNA Remote Management Utility programming RPQ enables you to have communication between a host System/370 (with OS/VS2 MVS) and a remotely located Series/1 system. The utility program executes on a Series/1 and allows a host system to manage the Series/1 system. The utility allows a Series/1 system to operate effectively without an operator and enables a remotely managed Series/1 based system to be a part of an all-SNA network.

A Virtual Telecommunications Access Method (VTAM) program is provided that allows the host system to initiate operator commands and send them to a Series/1 for processing. You can issue most operator commands (except those requiring operator intervention at a Series/1) as if the host terminal was directly connected to a Series/1 system.

Highlights of the utility include:

- Transmission of files to and from a Series/1 system
- Ability to perform storage dumps, execute programs, and transfer diagnostic information back to the managing system on demand
- SNA host connection

The host gives the Series/1 commands across a synchronous data link control (SDLC) connection. This capability allows a programmer at a Time Sharing Option (TSO) terminal on the host system to monitor and control the activity of a remote Series/1 system. The session between the System/370 and a remote Series/1 can be established using a point-to-point leased connection or using a multipoint or point-to-point dial-up connection.

Documentation

Complete information on using the SNA Remote Management Utility is located in the *SNA Remote Management Utility User's Guide.*

Series/1 to System/370 Channel Attach Program

The Series/1 to System/370 Channel Attach Program allows a Series/1 user to transfer data, under joint consent, between application programs executing on a Series/1 system and a host System/370, 30xx, or 43xx system. The host system must be using OS/VS1, OS/VS2 (SVS or MVS), or DOS/VSE with the Basic Telecommunications Access Method (BTAM).

The Channel Attach Program:

- Establishes, controls, and terminates access between Series/1 application programs and the channel attach device
- Transfers input and output between the Series/1 application programs and the channel attach device
- Logs errors
- Handles interrupts from the channel attach device
- Performs error recovery at the Realtime Programming System READ/WRITE access level

Documentation

Complete information on the channel attach program is located in the *IBM Series/1-System/370 Channel Attach Program General Information Manual* and in the *IBM Series/1-System/370 Channel Attach Program Reference Manual.*

3270 Device Emulation Using the Multiple Terminal Manager

In addition to its support for local transaction processing, the Multiple Terminal Manager licensed program enables you to physically connect a Series/1 system and a host System/370 system. This support is called 3270 device emulation. The device emulation uses a locally attached full-screen formatted terminal and a line or matrix printer.[3] The device emulation enables a Series/1 processor to appear to a host System/370 as one or more 3270 Information Display Systems. Any of the manager's full-screen formatted terminals are used in this environment. End users entering information at the terminal appear to the System/370 as 3277 or 3278 terminal users. At the same time, transaction applications that are not communicating with the host system are able to use other terminals attached to the Series/1 processor for local processing.

The Series/1 system supporting 3270 device emulation is attached to the host System/370 using either the Series/1 binary synchronous communications protocol or the synchronous data link control (SDLC) protocol.

If the binary synchronous communications protocol is used, the host system recognizes:

- The Series/1 system as a 3271 Model 2 control unit
- Display terminals in character mode as 3277 Model 2 display stations

If the synchronous data link control protocol is used, the host system recognizes:

- The Series/1 system as a 3274 control unit
- Display terminals as 3278 Model 2 display stations

When the SDLC protocol is used, an IBM 4973, 4974, 4975, 5219, 5224, or 5225 printer attached to a Series/1 using 3270 device emulation appears to the host system as a 3287/3289 printer using an SNA character string.

Documentation

Complete information on the 3270 Device Emulation support is included in the *Multiple Terminal Manager Version 3 User's Guide.*

[3] The binary synchronous communications protocol supports terminals only.

X.25/HDLC Communications Support

The X.25/HDLC Communications Support (XHCS) licensed program enables a Series/1 to communicate with devices connected to an X.25 packet switching network. The modules that comprise XHCS:

- Provide X.25 packet-level protocol management:
  - Supports multiple permanent virtual circuits or switched virtual circuits (virtual calls) on a single network access link
  - Supports data communications applications where Series/1 is a Data Terminal Equipment attached to an X.25 network, or where Series/1 is a node within an X.25 network
  - Supports a READ/WRITE/CONTROL assembler language macro interface to application programs
  - Provides access to carrier-supported optional facilities that are controlled by application programs (for example; reverse charging, closed user group, packet and window size negotiation, throughput class negotiation, and priority service)
- Provide HDLC (High-Level Data Link Control) protocol management:
  - Supports HDLC Asynchronous Balanced Mode protocols; that is, the LAPB protocol that is used to access X.25 networks
  - Supports HDLC Normal Response Mode protocols, point-to-point or multipoint
  - Supports data communications applications in which Series/1 communicates by means of one or more leased or dialed HDLC links
- Provide duplex and half-duplex data transmission at speeds of 1200, 2400, 4800, 9600, 19200, 48000, or 56000 bits per second

Documentation

For complete information see the *X.25/HDLC Communications Support: Programming and Operating Reference Manual.*

Remote Manager

The Remote Manager licensed program enables Series/1 networks to be managed and operated through the communications and systems management programs available on IBM host processors (System/370, 30xx, and 43xx). The Remote Manager on each Series/1 in the network supports centralized control and problem determination using the following host programs:

- Network Communications Control Facility
- Distributed Systems Executive
- Host Command Facility
- Network Problem Determination Application

The Remote Manager provides three major functions:

- Alert-processing facility that routes Series/1 hardware and software error indications to the Network Problem Determination Application at the host, alerting network operators to real or potential problems with the Series/1 network operations
- Host-operator facility that enables a host operator to act as a local Series/1 operator; issuing commands, invoking system utilities, and running application programs
- Relay and node data services facility that enables a Series/1 and a host to transmit data between one another using the Distributed Systems Executive

Documentation

For complete information see the *Remote Manager User's Guide.*

Communications Manager for the Series/1

The Communications Manager for the Series/1 manages communication:

- Between a Series/1 and other Series/1s
- Between a Series/1 and a host
- Between a Series/1 and various input/output devices

It supports a selected set of devices and communication connections and provides a structure for you to write support for other devices. The Communications Manager is also a message management program that manages the communication of messages to and from your applications.

Examples of applications that could effectively use the Communications Manager include:

- Data entry applications that accept messages from terminals. The application may process the data or merely collect the data and pass it along to another processor.

- Message routing applications that handle the routing and sending of messages within a network. A message routing application may, for example, send a message to all users or to a selected subset, or it may receive data from one processor and pass it along to another processor without analysis or processing.

- Applications that do a share of some data processing that is part of a larger system. Such an application may, for example, manage part of a data base or do preliminary processing of data, with the final processing being done in the host.

- Store and forward applications that collect messages arriving at a station, hold them, and then forward them to another location. An installation might, for example, forward the messages when a certain volume of them has been collected, at a certain time of day, or when communication line use is low.

- Line concentration applications that collect messages, batch them, and forward them efficiently over a high speed communications line

The Communications Manager includes a set of commands that enable a system operator or user to interact with the Communications Manager from a terminal. The commands enable you to:

- Start or stop the Communications Manager
- Define or delete stations and control their attributes
- Hold, release, and purge messages
- Display status information, error counts that pertain to a station, message queuing, and storage utilization information
- Remotely IPL a Series/1 system
- Load, unload, and locate specific modules
- Enter Realtime Programming System operator commands from within the Communications Manager environment
- Build and list the contents of an X.25 directory to establish X.25 connections
- Control message traffic over the Local Communications Controller ring with the Local Communications Controller program

You can define an error message logging station, a command logging station, and a user message logging station.

The Communications Manager can be installed either with the system utilities or with the command language facility. An installation can use Communications Manager functions without doing additional programming. However, the manager includes a set of assembler language macros as well as Cobol, PL/I, and Pascal callable routines for you to use in writing application programs in any of these languages. A Communications Manager subroutine makes it possible for Cobol, PL/I, and Pascal application programs to use Communications Manager functions to build and log their own error messages.

The Communications Manager also provides IBM 3271 control unit emulation. This feature enables a Series/1 attached to a host processor to serve as a 3271 control unit with up to 32 devices attached. The devices can be 3277 terminals or 3284/3286 printers. The Programmable Communications Subsystem is required for 3271 emulation.

Communications Manager 3270 support can be used to connect a Series/1 to an IBM 5280 Distributed Data System operating in 3270 emulation mode.

Message Handling

As a message management program, the Communications Manager controls the flow of messages between processors, devices, and application programs. A **message** is any unit of information—from a record of a sales transaction to a large file. The message managing facilities include:

- Delivering messages. The Communications Manager obtains a message from a processor, device, or application program and delivers it to another processor, device, or application program.
- Message priorities. The Communications Manager delivers messages on a priority basis. Message priority is assigned on the basis of the originating station's priority assignment during station definition.
- Queuing messages. The Communications Manager creates priority queues of messages for each destination either in processor storage or on disk.
- Message warm start. The Communications Manager delivers messages, contained in disk storage, that were not delivered during the previous execution of the manager.
- Device independence. The Communications Manager accepts a message in the communications configuration and delivers it to any location in the configuration. If the source and destination are devices, they need not be of the same device type.

Local Communications Controller

You can use a variety of communications protocols with the Communications Manager to send messages. For instance, the Communications Manager supports the Series/1 Local Communications Controller. This is a high-speed ring connection, which enables you to send data to any one of up to sixteen connected Series/1s.

The Communications Manager includes two message-path programs that support the Local Communications Controller but are mutually exclusive on the same Local Communications Controller. Both manage Series/1-to-Series/1 communication. One program, the multi-processor ring message path program, enables the Realtime Programming System's multi-processor support to be implemented on the ring. The other program, the Local Communications Controller message path program, enables the Communications Manager to communicate with another Series/1 at which either the Communications Manager or the Event Driven Executive's Communications Facility is running.

Communications Protocols

The Communications Manager also supports binary synchronous, SNA, or high-level data link control protocols for communicating with other processors. Binary synchronous, SNA, or channel attach protocols are used in communications with a host System/370. Binary synchronous communications support can also be used for a Series/1 system in communication with a host IBM System/3, System/23 Datamaster, Displaywriter, 6670 Information Distributor, 5280 Distributed Data System, and the 5520 Administrative System.

Using asynchronous (start/stop) ASCII communications, a Series/1 can communicate with the following IBM systems:

- Personal Computer
- System/23 Datamaster
- Displaywriter

The Series/1 system can also appear as a remote teletypewriter to a host system.

You can define your Communications Manager network so that your application programs are independent of the techniques used to send messages among applications.

Multiple Terminal Manager Bridge Service

The Multiple Terminal Manager combines its terminal services with the communications functions of the Communications Manager. This combination simplifies communication with remote systems in a transaction processing application.

Such a service could be used to communicate with other Series/1 systems or with an Information Management System (IMS) program, or with a Customer Information Control System (CICS) program running on a host System/370. You can communicate with the host subsystem(s) from any terminal that the manager supports.

X.25 Network Support

The Communications Manager provides two kinds of X.25 network support:

- The Communications Manager can send and receive X.25 data packets from other computers or devices that transmit or accept X.25 data packets through an X.25 packet-switching network
- Series/1s with the Communications Manager installed can be configured as an X.25 network. Other computers or devices that can send and receive X.25 data packets can communicate through this Communications Manager X.25 network

This support adheres to the CCITT Recommendation for X.25 protocol.

Communication with SNA Devices Through SDLC Pass-Through Support

The Communications Manager supports SNA devices attached to a Series/1 as secondary SDLC (synchronous data link control) devices. Through SDLC pass-through support, such devices as the 3274 control unit, the 4700 Finance Communication System, and the 3600 Finance Communication System can communicate with a host computer (such as a System/370) over one or more SDLC lines through a single Series/1 or through a network of Series/1s.

Documentation

For further information see the *Communications Manager for the Series/1 Introduction.*

Chapter 4. User Interfaces to the System

You communicate with the operating system through a user interface. A **user interface** is a means by which an individual uses a terminal interactively. User interfaces can be classified as:

- End user interface
- Programmer interface (for application programmer or systems programmer)
- Operator interface

An **end user** is someone who interacts with a program through a user terminal. A **programmer** develops and executes programs at a user terminal in a program development environment. An **operator** is anyone who controls system operation.

Contents of this chapter:

End User Interfaces

End user interfaces are developed by you or are supplied by IBM. A transaction-oriented application serving one or more terminals and allowing several users to enter transactions is an example of an end user interface.

The operating environment for an end user terminal is under the control of the application program that uses the terminal. Anything entered at the terminal is processed by the application. End user interfaces usually supply a set of commands or screen menus to start or stop the interface and to access interface functions. An end user development interface usually executes on a non-development (production) system.

Using the Command Language Facility

The command language facility is an IBM-supplied end user interface to system functions. The end user simply activates (logs on to) the command language facility, and the facility sets up a terminal session tailored for the end user. The end user uses the command language facility menus or enters commands directly at the terminal. The command language facility is discussed in greater detail in Chapter 5, "Development Tools and Application Aids" and in the *Command Language Facility User's Guide.*

Using the Multiple Terminal Manager

The Multiple Terminal Manager supports the operation of several different terminals that process application programs, from a single Realtime Programming System partition. User-callable routines enable programs to be written as conversations with an end user. The Multiple Terminal Manager is discussed in greater detail in Chapter 5, "Development Tools and Application Aids" and in the *Multiple Terminal Manager User's Guide.*

Programmer Interfaces

Programmer interfaces to various operating system functions include facilities such as the command language facility, the Communications Manager for the Series/1, a full-screen hexadecimal editor (FSEDIT), and a network definition utility.

Command Language Facility

The command language facility provides, via menus (which offer help information for the user) and commands, an operating environment unique to each programmer. This predefined environment reduces the amount of preplanning and system knowledge required to begin program development.

Command language facility commands provide commonly used functions that support either your development system or your production system. The command format minimizes the number of key strokes and provides command control via keyword parameters. Menus provide meaningful defaults when optional parameters are omitted. Together the menus and the command set supports the following functions:

- File creation and maintenance (including move, copy, rename, dump, print, delete, and file attribute listing functions)
- Creation and maintenance of data set definitions (DSDs)
- Set up, installation, or modification of a system using the configuration program; including displaying node configurations, naming nodes, designating task sets, allocating storage and copying custom-built systems to nodes
- Obtain information about another node in a multiprocessor system
- Copy S1/EXEC volumes and data set definition tables to other nodes in a multiprocessor system
- Specify internode communication control block allocations
- A command help facility

The menus and commands that support program development enable a programmer to compile, assemble, application build, and test programs written in PL/I, Cobol, Fortran, Pascal, or assembler language.

The command language facility is discussed in greater detail in Chapter 5, "Development Tools and Application Aids" and in the *Command Language Facility User's Guide.*

Communications Manager for the Series/1

The Communications Manager provides a programmer interface for controlling a communications environment in which many Series/1s and devices can be physically connected. The Communications Manager is discussed in greater detail in Chapter 3, "Data Communications and Network Support" and in the *Communications Manager for the Series/1 Introduction.*

Full-Screen Editor (FSEDIT)

FSEDIT is a full-screen editor for altering data in hexadecimal format. Blocks of data (up to 256 bytes) on disk or diskette are examined and changed, according to your instructions. You can save a block of data, cancel editing a block, print either a block or just the changes you have made, and find and/or replace a string (character or hexadecimal).

Network Definition Utility

With the network definition utility you can dynamically specify the configuration of an SNA network and the characteristics of SDLC devices. (This function could be done previously only during the SYSGEN process.)

Data can be entered by means of easy-to-use menus or by line commands, depending on your terminal type. You can define, modify, or delete data, or print a report about your SNA network or SDLC devices.

See the *Network Definition Utility User's Guide* for detailed information.

Operator Interface

The operator interface enables an operator to interact directly with the operating system to load a supervisor and control its operation using operator commands. Operator commands control the programs that execute on the system. These commands can be issued from an operator console, a remote terminal, or can be included in your program.

The **system command reader** interprets operator commands. The command reader accepts operator commands from any terminal that logs on to the terminal controller as an operator console. In a multiprocessor system, the command reader accepts commands from any node and performs the function. A command entered from a remote node is treated as if the command were entered at the local node. If errors occur on the operator console, the command reader tries to use an alternate terminal as the operator console.

In a multiprocessor system, an operator console need not be attached to the Series/1 you IPL. An operator console can be local or remote with all commands directed to one node or to separate nodes in a multiprocessor system.

See the chapters on "Operator Interface" and "Terminal Controller" in the *Supervisor Services Programming Guide* for a more detailed discussion of operator interface.

Installing the Standard System

The first task a support programmer performs is to install the system. IBM generates a standard system which is ready to be installed when you receive it. The standard system can be installed on a uniprocessor system or on each node of a multiprocessor configuration. Installing the Program Preparation Subsystem enables the standard system to support application development. Complete information on the standard system is provided in the *Standard System Installation Guide.*

Before the system can become productive, the support programmer must install the standard system on disk and must perform an IPL of the system. Typically, the support programmer:

1. Performs an IPL of the utility package called the stand-alone utilities. The stand-alone utilities reside on the stand-alone utilities diskettes shipped from IBM.
2. Copies the standard system libraries from diskettes onto disk using the system build utility
3. Copies the command language facility using the system build (SYSB) utility
4. Loads the standard system into processor storage
5. Installs other licensed programs (including programming RPQs) using the command language facility INSTALL command

The stand-alone utilities are discussed in more detail in Chapter 5, "Development Tools and Application Aids" and in the *Utilities Reference* manual.

Attended and Unattended Modes

A system is said to be operating in **attended mode** when an operator is present at the operator console. A development system usually runs in attended mode.

A production system can operate without an operator being present at the operator console. This mode is called **unattended mode**. Systems executing realtime applications usually run in unattended mode. In order to control system operation in a system executing in unattended mode, two different ways of executing operator functions and interactive operator commands are available:

- Operator commands are stored in the two special disk files that are executed when the system is loaded and started (IPL). These special files are the IPL options file and the user input command (UIC) file.
- Operator functions are started by executing system macro instructions in a program. The system macro instruction invokes the operator function.

In unattended mode, the system does not write messages to the operator console that require operator response or intervention because an operator is not present at the console. If a system error occurs in unattended mode, the error message is written to the system error log file.

Operator Command Set

IBM supplies a variety of operator commands for controlling both uniprocessor and multiprocessor systems. Most commands have an optional parameter (NODE=) that specifies which node of a multiprocessor system should execute the command. These commands include:

- A help command that displays the syntax and functions of operator commands
- Commands to control task sets and system task set tables:
  - Start a task set
  - Stop a task set abnormally
  - Update the system scheduler table so a task set is scheduled for a specific time of day, regular time-of-day intervals, or upon the occurrence of a process interrupt
  - Update the system task set table
- Commands to display status information for:
  - Active devices (including disk seek statistics)
  - Active programs (task sets)
  - Active storage partitions
  - Active tasks
  - Attached devices
  - Available processor storage
  - Messages to which the operator has not yet replied
  - Programs currently scheduled for execution by the system scheduler
  - Programs whose disk location is known to the system scheduler
  - Spooled jobs and spool writers
  - Status of nodes
  - Time of day and date

- Commands to control and monitor nodes in a multiprocessor system:
  - Abnormally stop a node
  - Display status of nodes
  - Start a node
  - Start Local Communications Controller support backup
  - Synchronize time and date
- Commands to manage devices:
  - Activate or deactivate an SNA network
  - Activate or deactivate support for a device
  - Assign a logical device number to a device
  - Change the setting of the Programmable Two-Channel Switch
  - Make a device ready for use
  - Mount or demount a diskette
  - Restore an entire disk (or logical volume) from diskettes/tapes
  - Save an entire disk (or logical volume) to diskettes/tapes
  - Switch a device status from offline to online or vice versa
  - Switch processing between a primary device and an alternate device
- Commands to debug system problems and handle errors:
  - Define and initialize an error log
  - Deactivate and free an error log
  - Delete the debug work area
  - Display free storage in a partition
  - Display logical storage for address spaces
  - Display size and address of the I/O trace log area
  - Display specific areas of storage on the operator console
  - Display status of a started device or all devices
  - Display system scheduler table entries
  - Display system task set table entries
  - Display task sets queued to be run
  - Dump processor storage to diskette
  - Dump the SVC data log
  - Load transient program
  - Make a data set available for dumping after its contents have been printed or copied
  - Patch storage
  - Resume I/O tracing across the system
  - Set operator mode to attended or unattended
  - Start a debug task
  - Start I/O tracing for a specific device
  - Start SVC tracing
  - Start transient monitoring
  - Stop the monitoring of transient programs
  - Suspend I/O tracing across the system
  - Unload a transient program
- Commands to gather disk seek statistics:
  - Display disk seek statistics
  - Reset counters in disk seek statistic tables
  - Start or stop the gathering of disk seek statistics

- Commands to control spooling to printers:
  - Delete a spool job
  - Expedite a spool job
  - Hold or release a spool job
  - Restart a spool writer
  - Start or stop a spool writer
  - Start or stop disk spooling
  - Split and print a spool job
- Initialization commands:
  - Define partitions during system initialization. DEFP is invoked from the IPL input file and cannot be entered at the operator console.
  - Perform an initial program load of the system
  - Set the system date
    Set the time of day
  - Stop the processing of the IPL input file. The STOP command is invoked from the IPL input file and cannot be entered at the operator console.
- Commands to define terminals and subsystems to the terminal controller:
  - Add or delete a terminal
  - Define a subsystem
  - Display the status of subsystems or terminals known to the terminal controller
  - Prepare a subsystem
  - Run a subsystem
- Commands to manage timers:
  - Set system date format
  - Set system time format
  - Synchronize timers in all nodes in a multiprocessor system
- Commands to monitor performance:
  - Start/stop/reset/log
- Commands to control and monitor the internode communications facility:
  - Display internode communications facility statistics
  - Display internode communications facility trace table entries
  - Reset the internode communications facility statistics
  - Start internode communications facility tracing
  - Start the gathering of internode communications facility statistics
  - Stop internode communications facility tracing
  - Stop the gathering of internode communications facility statistics
- A command, OPTS, to set system options

Complete information on operator commands is found in the *Operation Guide and Reference* manual.

Interactive Subsystems

Programs that support transaction processing or interactive program development are called **interactive subsystems**. Interactive subsystems manage terminals and service programs in an orderly fashion. The terminal controller component of the operating system provides many of the facilities necessary to support interactive subsystems and their terminal operating environments.

Subsystem Connection

The terminal controller provides a single easy-to-use command that allows you to connect a terminal to an interactive subsystem. This logical connection is accomplished by entering the LOGON command followed by the name of the requested subsystem.

Complete information on the terminal controller is in the *Supervisor Services Programming Guide*. Complete information on the terminal controller commands is in the *Operation Guide and Reference* manual.

Subsystem Definition

The operating system allows both IBM subsystems and subsystems written by you to be defined to the terminal controller. A subsystem is defined via an operator command (LOGON CONSOLE) or by a program using the system macro instruction for defining subsystems. A subsystem definition provides the operating system with the following kinds of information:

- Single-user versus multiple-user subsystems — whether one copy of the subsystem is for a single user or for multiple users. A single-user subsystem requires one copy of a subsystem for each active terminal. A multiple-user subsystem allows multiple terminals to share a single copy of a subsystem.
- The number of the storage partition(s) in which the subsystem can execute
- The names of programs that are executed when a single-user subsystem is activated for the first time. For all subsystems, the individual defining the subsystem can indicate which program within a subsystem is executed when a LOGON command activates a subsystem.

Complete information on defining and preparing subsystems is in the *Supervisor Services Programming Guide.*

Controlling Terminals

In many subsystem environments, you may want to connect a given terminal to different subsystems at different times. Also, you must be able to recover terminals from a subsystem whose execution terminates abnormally. The terminal controller allows this level of control by providing:

- A terminal definition command that allows an operator to specify which interactive terminals are to be assigned to the terminal controller. Only those terminals that are assigned to the terminal controller can be logically connected to subsystems via the LOGON command.
- A facility for recovering interactive terminals when a subsystem terminates. This facility ensures that the terminal controller regains ownership of all terminals assigned to it.
- A CANCEL command that allows anyone using the subsystem to stop the execution of the program that is currently executing under the control of the subsystem. This facility permits you to re-establish a terminal dialog with the subsystem.

Chapter 5. Development Tools and Application Aids

The Realtime Programming System supports:

- Programming languages and subroutine libraries for program development
- Interactive tools for development and production systems
- Non-interactive (batch processing) tools for development and production systems
- A facility for preparing executable programs (task sets)
- Application aids that extend the capabilities of the operating system

Contents of this chapter:

Programming Languages and Subroutine Libraries

A long-established concept in data processing is that high-level programming languages increase application programmer productivity. Increased productivity is achieved through the use of generic statements, debugging aids, and tools for modifying the program. Four high-level programming languages are offered on the Series/1 under the Realtime Programming System:

- Cobol
- Fortran
- Pascal Programming RPQ
- PL/I

Highlights of each language are presented in the following sections. If you need details on any of the languages, see the individual reference manual for each language.

Cobol

Series/1 Cobol is a programming language for developing commercial and online data entry applications. Cobol is suited for an interactive commercial environment where users run transaction processing applications such as order entry and data collection as well as applications that run in a background batch environment. Series/1 Cobol supports input/output functions that enable you to accept data from, and display data to, a terminal.

Cobol generates executable code for Cobol verbs and calls library subroutines that interact with the operating system and perform complex data handling. The language offers a wide range of commercial features, plus facilities for handling input and output, sorting and merging data files, structuring the source and object programs, and debugging Cobol programs.

In addition to facilities for accepting data from, and displaying it at Series/1 interactive devices, Series/1 Cobol also supports a variety of debugging and productivity aids.

The Series/1 Cobol compiler with a resident subroutine library translates Cobol source programs into executable machine instructions and data.

The transient subroutine library consists of subroutines that are loaded on demand to perform various functions for the compiled code during execution. The transient library is required on machines where Cobol programs are executed but is not required on those machines where only compilation is done.

The two Cobol libraries (resident and transient) handle:

- Arithmetic conversion
- Decimal and binary arithmetic
- Transfer of data between data areas
- Error handling and message processing
- Sort/Merge interface
- Input from and output to magnetic tape
- File processing (communication between the object program and the operating system I/O routines)

Series/1 Cobol supports the American National Standard (ANS) X3.23-1974 as understood and interpreted by IBM as of March 1979, with the exception of the RERUN clause. The particular subset of the language is characterized as the low-intermediate level, as defined in FIPS[1] Publication 21-1, with some additional ANS features.

[1] Federal Information Processing Standard

Highlights of Series/1 Cobol are:

- Table processing with up to three dimensions
- Sorting and merging of up to eight data files
- Program segmentation enabling you to divide the Procedure Division into overlay segments
- Configuration, Input-Output, and Procedure Section paragraphs and file description (FD), sort description (SD), and record description (RD) entries are copied from a library into a source program
- Interprogram communication by calling other Cobol program modules or by calling program modules previously written in assembler language, Fortran, or PL/I
- Sequential, relative, and indexed file processing, including fixed block relative support. Fixed block relative support provides faster processing of relative files and better utilization of direct access storage by permitting short-length records to be blocked.
- Debugging features and productivity aids that include:
  - Compile-time storage maps
  - Execution time "snapshots" of data areas
  - Flow trace which identifies the last statement that was executed before a program module terminated abnormally
  - Error checking
  - Error messages at five severity levels
  - Identification of source statements that exceed the specified FIPS language level
  - Options to control the form of the compiler output listing
- Callable subroutines that provide an interface to the operating system services of:
  - Loadable external modules
  - Global queues
- Magnetic tape support

Documentation

Complete information on using Cobol is in the *Cobol Language Reference* manual and the *Cobol Programmer's Guide.*

Fortran

Series/1 Fortran is a high-level programming language that is primarily used for mathematical applications. Fortran has also been used for other data processing applications such as those to perform data reduction or that produce printed reports. To fully support the I/O needs of a variety of applications, Series/1 Fortran supports direct access files and formatted and unformatted I/O.

Series/1 Fortran requires the Mathematical and Functional Subroutine Library. Fortran, with the required Mathematical and Functional Subroutine Library, is a subset of American National Standard Fortran (X3.9-1966). Fortran also meets the Instrument Society of America (ISA) S61.1 1976 standards for executive functions, process input/output, and time and date functions.

Fortran also requires floating-point support (either the floating-point emulator or the floating-point hardware feature).

Series/1 Fortran also provides a set of callable routines in the optional Realtime Subroutine Library. These routines support realtime application environments. The capabilities provided by the Reáltime Subroutine Library are described in the section entitled "Realtime Subroutine Library" on page 5-7.

Highlights of Series/1 Fortran are:

- Logical and relational operations for testing conditions
- Direct access input and output
- Formatted and unformatted input and output
- Execution flow tracing and dynamic debugging statements
- Execution-time error exits
- Subroutines for performing decimal arithmetic operations
- Mathematical built-in functions
- User programs optimized for storage efficiency and speed
- Facilities for dividing programs into logical units including subroutine subprograms, function subprograms, and arithmetic statement functions
- Callable subroutines that provide an interface to the operating system services of global queues and loadable external modules

Documentation

Information on using Series/1 Fortran is in the *Fortran Language Reference* manual and the *Fortran User's Guide.*

Mathematical and Functional Subroutine Library

The Mathematical and Functional Subroutine Library (MFSL) is a set of subroutines that can be used by application programs written in Fortran or assembler language. The types of MFSL subroutines are:

- Mathematical (including the following functions: logarithmic, exponential, trigonometric, minimum value, maximum value, modular arithmetic, positive difference, and transfer of sign)
- Conversion (including conversion from numeric EBCDIC format to an internal representation in integer or floating-point format or vice versa)
- Error checking and recovery routines
- Service routines for library work area initialization, work area termination, and abnormal termination
- Commercial routines that perform formatted output editing, data conversion, variable-length decimal arithmetic, and frequently needed utility functions such as moving data between areas in storage

The assembler language programmer can use the subroutines in the Mathematical and Functional Subroutine Library through the CALL macro instruction. A parameter list for the CALL macro passes a list of arguments to the subroutine being called.

The Fortran programmer executes some MFSL subroutines as Fortran FUNCTION subprograms, some subroutines by Fortran CALL statements, and other subroutines implicitly in Fortran statements.

Documentation

Complete information on using the Mathematical and Functional Subroutine Library is in the *Mathematical and Functional Subroutine Library User's Guide.*

Realtime Subroutine Library

The Realtime Subroutine Library is used with Fortran programs that require realtime functions. The library enables you to manage resources and control the execution of programs in a multitasking environment. The library includes routines to manage programs (task sets) and tasks, to manage partitions, to synchronize the use of global areas shared between programs, and to manage queues, interrupts, events, and serially reusable resources.

The Instrument Society of America (ISA) executive function enables you to start the execution of a program (task set) after a time delay or at a particular time of day and to place a task in a wait state.

The ISA process routines enable you to use sensor I/O devices. The routines enable reading from or writing to analog and digital points.

Documentation

Complete information on using the Realtime Subroutine Library is located in the *Fortran User's Guide.*

Pascal

The Pascal language covers a wide range of applications such as business data processing, scientific and general problem solving, and industrial automation. Pascal encourages the programmer to use structured programming techniques. The language includes many data types and enables user-defined data types.

Series/1 Pascal input/output features include sequential and random files, display to the users terminal, formatting facilities and many input/output extensions to standard Pascal.

The Series/1 Pascal compiler offers debugging aids (such as compiler diagnostics comparable to Pascal/VS, execution messages, and a trace function) and reentrant object code. Series/1 Pascal programs can be organized into overlays.

Series/1 Pascal:

- Provides constructs for defining data structures in a clear manner which resembles natural language
- Is a relatively machine-independent language
- Generates efficient object code and permits extensive error diagnostics during compilation
- Is suitable for applying structured programming techniques
- Enables extensive execution-time checks for programs

Series/1 Pascal is compatible with the proposed International Standards Organization (ISO) standard (as of June 1981) with extensions, and is a compatible subset of Pascal/VS.

Documentation

Complete information on using Pascal is in the *Pascal Language Reference* manual and in the *Pascal User's Guide*.

PL/I

Series/1 PL/I is a powerful general-purpose language. PL/I is effective in the following programming areas:

- Transaction processing
- Problem solving
- Scientific applications
- Traditional data processing applications
- List processing
- Realtime applications (such as plant and laboratory automation, process control, or sensor-based applications)

Series/1 PL/I is a subset of the American National Standards Institute PL/I (ANSI X3.53-1976), with language extensions added to address areas such as realtime processing. PL/I gives you access to most operating system functions.

The PL/I licensed program consists of a compiler and a resident library. The resident library contains frequently used routines. Another licensed program, the transient library, is required to build and execute a PL/I produced application program. The transient library contains less frequently used routines such as I/O transmission, error handling, and conversion routines. Transient library routines are loaded dynamically when needed during execution and thereby conserve storage use.

Highlights of Series/1 PL/I are:

- Error detection and debugging aids to decrease program development time:
  - Compile-time diagnostics
  - Execution-time diagnostic messages
  - ON-unit support to control error conditions
- Language extensions that enable a programmer to:
  - Schedule, execute, and control external procedures as independent tasks. This capability is restricted to the local node.
  - Schedule, execute, and control the execution of other PL/I programs. This capability is restricted to the local node.
  - Respond to and control events associated with time of day, process interruptions, task termination, I/O control, and other user defined actions
- Input/output capabilities including:
  - Formatted and list-directed stream I/O
  - Consecutive input/output
  - Indexed or random file access by keys using REGIONAL or INDEXED access. Indexed access uses the Indexed Access Method licensed program.
  - Sorting and merging of files using the Sort/Merge licensed program
  - Binary synchronous and start/stop data communications using record I/O statements
  - Magnetic tape support
  - Record I/O support for sensor I/O
  - Record I/O support for display terminals
- Multiple data types including:
  - Arithmetic data in either binary or decimal radix, fixed or floating scale
  - String data in either character or bit form, with fixed or varying attributes
  - Picture data in character or numeric field formats
  - Arrays (with up to 15 dimensions) and structures

- Data manipulation features including:
  - String and Boolean operations
  - Built-in functions (string and mathematical)
  - Array and structure expressions
  - Automatic data conversions in expressions
  - Support for internal and external subroutine and function procedures
- Object time efficiency:
  - Reentrant generated code
  - Object-code compiler optimization feature
- Built-in subroutines to enable access to special operating system functions:
  - Dump storage
  - Invoke any system function via supervisor call (SVC)
  - Set the time and date (only if this function is included when the system is generated)
- Callable subroutines that provide an interface to the operating system:
  - Storage services
  - Loadable external modules
  - Global queues

Documentation

Complete information on using Series/1 PL/I is in the *PL/I Language Reference* manual and the *PL/I User's Guide*.

Macro Assembler

For some types of applications, typically those which extend the facilities of the operating system, it is necessary to gain high program efficiency. To gain this efficiency, a programmer uses the Series/1 macro assembler and writes programs in Series/1 assembler language.

The macro assembler enables a programmer to write Series/1 machine code using symbolic operands for data areas, program names, and instruction labels. In addition to these facilities, the programmer can cause the assembler to generate sequences of instructions on encountering pseudo-instructions that are programmer-defined. These pseudo-instructions are called macro instructions and are defined to the assembler in macro language.

IBM provides a set of macro instructions for the Realtime Programming System. These macros are used in assembler language programs to request system services. They are also used in the subroutine libraries of the high-level languages.

In addition to the macros provided to request system services, IBM supplies a set of structured programming macro instructions that are aimed at making assembler language program development and maintenance easier. These macros contain a great many useful programming aids such as:

- Block definition (BLOCK statement)
- Loop definition (DO statement)
- Case selection (CASE statement)
- Comparison operations and control (IF, THEN, ELSE statements)
- Exit a structure (LEAVE statement)

Documentation

Necessary manuals for using the Series/1 macro assembler include the *Macro Assembler User's Guide*, the *Macro Reference* manual, and the *Messages and Codes* manual. Summary information is located in the *Reference Summary.*

System/370 Host Preparation Facilities Programming RPQ

The System/370 Host Preparation Facilities Programming RPQ is a language translator that translates symbolic instructions into Series/1 machine language instructions, assigns storage locations, and performs auxiliary functions necessary to produce executable machine language programs. These facilities are language comparable with the native Series/1 Assembler.

The host preparation facilities are not a part of the Realtime Programming System. IBM provides this separately-priced licensed program for use on System/370 systems that use the OS/VS2 MVS operating system.

The host facilities enable you to use program development tools on the System/370 such as SPF and TSO. The host tools also enable Series/1 programming groups to maintain user program libraries and develop new applications in a controlled fashion. The host facilities:

- Produce object modules suitable for input to the System/370 Host Application Builder
- Enable user macros
- Provide conditional assembly instructions
- Provide printer output including program listing, symbol dictionaries, cross-reference and diagnostic messages

To complement the System/370 Host Assembler, a macro library of structured macros is provided for use as an aid in writing assembler language programs.

System/370 Host Application Builder

The System/370 Host Application Builder combines separately-assembled object modules into an executable load module or task set. This facility is comparable to the Series/1 application builder and performs the following functions:

- Create task sets for execution under the Series/1 Realtime Programming System Version 6
- Create load modules for user's own control program
- Aid in construction of overlay program segments
- Aid in program modification by replacing program segments
- Produce storage map and error listing on the printer

System/370 Environment

The System/370 Host Assembler programming RPQ runs in an environment provided by System/370 OS/VS2 (Multiple Virtual System) Control Program.

After the application task sets or load modules have been created on the host processor, it is necessary to transfer the data sets containing these modules to the Series/1 for execution. The Remote Manager licensed program provides a means for transmitting data between a System/370 and Series/1.

Additional Prerequisites

The following licensed programs are prerequisites for this programming RPQ.

- *Series/1 Realtime Programming System Version 6* (Program number 5719-PC6) for executing programs on the Series/1 that were built using the host facilities, and for providing the environment to transmit data between the Series/1 and System/370
- *Series/1 Program Preparation Subsystem Version 6* (Program number 5719-AS6) for obtaining Realtime Programming System macros that can be transmitted to the System/370 for use with the Host Assembler

Interactive Tools for Development and Production Systems

Interactive tools for development and production systems include the command language facility (including the S1/EXEC language), the text editor, and the utilities.

Command Language Facility

The command language facility is the interactive interface to operating system functions and to development languages that are installed on your system. The command language facility runs as a subsystem and is aimed primarily at a multiple terminal environment. This subsystem defines a user environment consisting of a set of files for the retention of source programs, compiler output files, and executable programs. The existence of these files gives you a set of defaults that the subsystem uses and thereby eliminates the need to specify a great many detailed parameters. You can compile, edit, and save a program with a minimum number of key strokes.

You can use the command language facility either directly or through a set of menus. If you choose to use the menus, you will be able to use the help screens that accompany them. The help screens provide you with tutorial information about the various functions the options perform.

The S1/EXEC interpreter provides the interactive interface of the command language facility. The interpreter executes command files written in the S1/EXEC language. (This language is called S1/EXEC because it is a Series/1 adaptation of EXEC languages already in use on the Conversational Monitor System component of the VM/370 operating system.) Using the S1/EXEC language, you can write commands in English-like words in free format. These commands are stored in a command file that is executed by the S1/EXEC interpreter. You request the execution of a command file by entering the name of the command file at your terminal. You can also write your own menus and the command files to process them.

IBM supplies a set of command files that are aids in the program development area. These commands support functions to:

- Create and edit source and data files
- Install other IBM programs, such as compilers
- Create, delete, or copy files
- Define files to the system
- Execute programs
- Ask for help about available commands and functions
- Compile programs in assembler language, PL/I, Fortran, Cobol, or Pascal
- Obtain information about another node in a multiprocessor system
- Copy S1/EXEC volumes and data set definition tables to other nodes in a multiprocessor system

Documentation

See the *Command Language Facility User's Guide*. The *Reference Summary* provides summary information.

S1/EXEC Language

In addition to the functions generally performed by a command language, S1/EXEC is a high-level programming language. S1/EXEC can be used to solve some traditional data processing problems. For example, a simple EXEC command file to generate a report could be written entirely in the S1/EXEC language.

Highlights of the S1/EXEC language are:

- Input and output statements for disk and diskette files
- Input and output to your terminal
- Ability to define variables and assign values
- Ability to pass data from an executing command to another command file
- Statements for looping, branching, conditional execution, data manipulation, integer arithmetic, and execution-flow tracing
- Recursive execution of a command file
- File creation, definition, and deletion

Documentation

The necessary manual for using the S1/EXEC language is the *Command Language Facility User's Guide*. The *Reference Summary* provides summary information.

Text Editor

You enter and maintain source programs and data under the Realtime Programming System using the text editor, a component of the Program Preparation Subsystem. The text editor is patterned after the editing facilities of the System Productivity Facility (SPF) offered on larger IBM systems.

The text editor supports three editing modes:

- Full screen (visual)
- Single line
- Non-interactive

The full screen and single line modes are interactive and depend upon a dialog between an individual sitting at a terminal and the edit program. Interactive mode is typically used in a program development environment.

The non-interactive mode is provided so that a program can build a file of edit commands that are used to control the editing process.

You invoke the editor via the command language facility EDIT command. You can specify the file you want to edit on the EDIT command line. If you do not specify a file name, the text editor displays a file selection menu (or prompts you to choose a file to edit). If you select a file that does not exist, the text editor creates the file for you.

Full-Screen Editing

The text editor is designed to make it easy to enter and modify data. In full-screen mode, you can work with the entire display screen and modify any portion of the data by overtyping existing characters on the screen. Pressing the ENTER key causes the full screen to be read and stored by the editor. Further highlights of interactive editing are:

- Line commands to:
  - Insert one or more lines for data entry
  - Delete one or more lines
  - Duplicate one or more lines, one or more times
  - Shift data on one or more lines
  - Display an indicator line to aid in aligning columns
  - Copy one or more lines to another place within a file
  - Move one or more lines from one place to another in a file
- Primary commands to:
  - Find text strings
  - Change text strings
  - Save the data being edited
  - Cancel the edit session without saving the edited data
  - Copy an external file into the current file
  - Control the translation of data to upper case
  - Print the contents of the edited file
  - Scroll the data being edited in any direction and any amount
  - Set tabs for high-level language source entry
- Program function key tailoring—This means you can equate the program function keys to any of the primary commands (with any applicable operands) so that the specified command is executed whenever the key is pressed

Single-Line Editing

Single-line editing means that the editor recognizes one line of a file as the current line on which the editing commands will have an effect. Editing commands are available to position the editor's line pointer to any line of a file, thus making that line the current line.

Non-interactive Editing

The text editor executes in non-interactive mode when editor commands and any new data are entered from a file on disk or diskette rather than from an operator keyboard. In non-interactive mode, the editor provides editing commands to:

- Change the position of the current line pointer
- Change text strings
- Find text strings
- Save the data being edited
- Cancel the edit session without saving the edited data
- Copy an external file into the file being edited
- Copy one or more lines to another place in a file
- Control the translation of data to upper case
- Insert one or more new lines
- Delete one or more lines
- Move one or more lines from one place to another in a file

Commands also exist that enable you to select the data to be edited if no file name is specified.

The text editor does not use absolute, fixed line numbers in editing. You refer to lines by context or by their relative position, either compared to the current line or compared to the top of the file.

Documentation

The necessary manual for using the text editor is the *Text Editor User's Guide*. The *Reference Summary* provides summary information.

System Utilities

The system utilities program, SYSUTILS, performs system maintenance and utility functions. System utility commands are entered interactively from a terminal or non-interactively from a file. Many command language facility commands invoke system utilities indirectly in non-interactive mode. The SYSUTILS program can be executed interactively under the command language facility by issuing the RUN command. Program function keys 1 to 5 can be tailored for use with the system utilities. Program function key 1 (PF1) is the default.

Eight utilities are provided under SYSUTILS, some of which have multiple options called subfunctions:

- The COMPRESS utility consolidates all available free space into one contiguous area within the specified system format device or file to reclaim unused (fragmented) space
- The COPY utility transfers data from one location to another on the same device or different devices. You can:
  - Copy an entire device or file as a unit with the options of compressing or backing up the file
  - Add records to the end of a file
  - Copy a single file or multiple files
  - Copy the entire physical contents of one diskette to another diskette
- The DEFINE utility is a multifunction utility that defines a disk or diskette to be in system format, creates a file, renames a file, deletes a single file or multiple files, and builds or deletes a data set definition (DSD)
- The INITIALIZE utility formats a diskette in either basic exchange format or extended format, checks and flags defective cylinders, and initializes a volume label and header labels
- The IPLMAINT utility:
  - Prepares a diskette used to perform an IPL of the processor storage-to-diskette dump utility
  - Assigns the name of the system to be loaded by the next IPL sequence
  - Installs a system task set for eventual IPL
- The MERGE utility combines two files or system format devices into a third file or system format device. After the merge, you still have the two original files plus the third file that contains the combined files.
- The PATCH utility modifies information stored on a disk or diskette. You have the option of verifying the existing data before the patch is applied.
- The REPORT utility has several subfunctions. Output from several of these subfunctions can be scrolled if it is directed to a display terminal. The REPORT utility generates the following listings:
  - A directory or table of contents list for a logical device, volume, or file
  - A dump of data contained in a device, volume, data set, or member
  - A printout of a previously generated processor storage dump
  - A listing of the data set definitions in the node data set definition table
  - A report of the contents of the task set reference table (TSRT) of a task set
  - A listing of the diskettes contained in the 4966 Diskette Magazine
  - A report of the system error log including device error analysis reports and summaries of all the error records contained in the log

Documentation

See the *Utilities Reference* manual. The *Reference Summary* provides summary information.

Tape Utilities

The tape utilities program, TAPUTILS, performs magnetic tape maintenance and utility functions. Tape utility commands are entered interactively from a terminal or non-interactively from a file. Some command language facility commands invoke tape utilities indirectly in non-interactive mode. The TAPUTILS program can be executed interactively under the command language facility by issuing the RUN command.

The tape utilities provide the following types of functions:

- COPY,TAPE copies data from a tape on one drive to a tape on another drive
- COPY,BACKUP copies data for backup from disk to tape, or from tape to disk
- DEFINE,BUILDTP builds a data set definition (DSD) for a magnetic tape
- INIT,TAPE initializes a tape for use with the operating system, or exercises a tape and tape device

Documentation

See the *Utilities Reference* manual. The *Reference Summary* provides summary information.

Stand-Alone Utilities

Stand-alone utilities—Assign, Disk Initialization, Report Directory, Report TSRT, Restore, Save, System Build, Tape Initialization, and Processor Storage-to-Diskette Dump—reside on diskette.

All but the last of the above utilities execute under a special diskette-based operating system. The utilities and the diskette-based system reside on the first three diskettes in the group of diskettes you receive from IBM. To invoke these utilities, you IPL the appropriate diskette and enter the appropriate command.

Note: The diskette device must be physically attached to the node where the IPL is being performed.

The Processor Storage-to-Diskette Dump Utility resides alone on another diskette. It is invoked by a manual IPL from the dump diskette containing this utility.

The stand-alone utilities provide the following types of function:

- The Assign Utility assigns the name of the system to be loaded by the next IPL sequence
- The Disk Initialization Utility initializes your 4962, 4963, 4967, or integrated system disk (4954-30D, 4956-30D, or 4965-30D)
- The Report Directory Utility provides you with a listing of entries in a system format device or exchange data sets
- The Report TSRT Utility produces a formatted report of a task set reference table data set, which includes each module name, its starting address, length, and alternate entry points
- The Restore Utility restores an entire disk device or logical volume from diskette or tape
- The Save Utility saves an entire disk device or logical volume to diskette or tape
- The System Build Utility installs an operating system on your 4962, 4963, 4967, or integrated system disk (4954-30D, 4956-30D, or 4965-30D) It can be run at any time to restore your IBM-supplied system, system utilities, and command language facility files.
- The Tape Initialization Utility initializes tapes mounted on your 4968 or 4969 tape unit
- The Processor Storage-to-Diskette Dump Utility dumps the contents of processor storage and registers. This dump may be used as part of the material submitted with an APAR, or as input to the REPORT,STGDUMP system utility.

Documentation

See the *Utilities Reference* manual. The *Reference Summary* provides summary information.

Batch Processing

Programs that execute in background mode are usually long executing, low priority programs that do not require interaction with a terminal. These programs are queued (batched) and executed one after another. **Batch processing** means executing a set of programs that are stacked or batched together. A batch of programs to be executed is collectively called a job. Typically, no human intervention occurs during the execution of programs in a batch job. Batch processing is available through the job stream processor component of the Program Preparation Subsystem. The Job Stream Processor Programming RPQ is available for batch processing applications executing on production systems that do not have the Program Preparation Subsystem.

Job Control Language

The job stream processor is a program that supplies a job control language for batch processing. The job stream processor reads and processes a job stream. A job stream is composed of statements requesting program execution, along with the file definitions needed for program execution. Job streams are created using command language facility commands that:

- Invoke the text editor to create the file containing the job stream
- Submit the job stream for execution

A batch job is executed in either foreground or background mode. In background mode, a batch job is executed when you transmit (submit) the job stream file to the job stream processor for execution using the command language facility SUB command. In foreground mode, the job stream processor is executed at an operator console using the TSET STR operator command.[2] Batch jobs are usually executed in background mode because they are typically long executing jobs with low priority.

Once started, the job stream processor reads the statements that comprise the job stream file. If the job stream processor executes in foreground mode, the source of the job stream could be your terminal, a disk or diskette file, or a combination of both.

An input stream is logically divided into independent units of work called jobs. The input stream can consist of one or more jobs that are processed sequentially. Jobs are self-contained elements and typically have no relation to other jobs in the input stream.

Each job can be composed of steps, which are basic units of work for the job stream processor. Each step executes a program (task set).

[2] The job stream processor cannot run in command language facility partitions. It executes in a separate partition (usually termed the batch partition).

Job stream processor statements enable you to:

- Redirect the source of the input stream of the job stream processor. You can direct the job stream processor to read the job stream from one device and then redirect it to read from another device.
- Define files and devices that your executable program requires
- Indicate the end of the input stream
- End a batch job session
- End a step
- Execute a program (task set)
- Denote the beginning of a job
- Prevent the execution of a step (used primarily if the input stream is from an interactive device)
- Pass execution-time parameters to the program (task set) being executed
- Indicate the end of data being spooled from an interactive terminal to a file

Additional job stream processor statements are available for batch processing on a development system. These additional statements provide a means of compiling, application building, loading, and executing source programs written in PL/I, Pascal, Fortran, and Cobol.

Documentation

To use the job stream processor see the *Batch User's Guide.* Summary information is in the *Reference Summary.*

Aids for Transaction Applications

A **transaction application** is a program whose execution is directed based on responses that a user makes at a terminal. A **transaction processing application** consists of a set of programs, data files used by the programs, and display formats on a terminal screen. A transaction application is interactive and usually enables several end users at different terminals to enter transactions concurrently.

Typically, the end user activates the transaction application, enters data in fields on a menu screen, and releases the screen as input to the transaction application by pressing the appropriate terminal key. The transaction program then processes the input screen and perhaps uses the information to invoke another application or to update a data base.

In general, transaction applications enable you to:

- Invoke a transaction from a local or remote terminal
- Perform inquiries and updates to a data base (random or indexed)
- Communicate with remote Series/1 systems

Aids to help you with transaction applications are:

- Multiple Terminal Manager
- Query

Writing Transaction Applications with the Multiple Terminal Manager

The Multiple Terminal Manager (or simply manager) is a licensed program that can be used with the Realtime Programming System to write transaction applications. The routines can be written in assembler language or in a high-level language (Cobol, PL/I, Fortran, or Pascal).

Multiple Terminal Manager Capabilities

Device Independence: The manager provides a high degree of terminal device independence for developing transaction processing applications.

I/O Interfaces: The manager supplies a set of I/O interfaces that you use when writing a transaction application. The manager requires that your programs, screen formats, and other parts of your transaction processing application be in specific files. These files are:

- A partitioned data set containing the transaction programs of your transaction application
- A file containing the screen formats used by your application
- A file to control who can sign on and use the transaction application (optional)
- Data files

  The manager opens and closes all files for you.

Callable Routines: Callable routines that control the logic flow within the transaction application enable you to:

- Start the prompt-reply process
- Temporarily suspend execution so another program can execute. You write your application so that processing time is shared among multiple users.
- Pass control from one application program to another program, or multiple levels of programs, and receive control at their completion
- Link to another routine from a currently executing routine

Callable routines that communicate with IBM display terminals include the capability to sound the terminal alarm, blink the cursor, obtain and display a screen, write output to the screen, right-justify a field, and enable or disable program function keys.

A callable routine called FILEIO handles all disk file I/O for the transaction application.

The manager also enables you to control what routines execute at which terminals or to control end user access to certain routines. The manager supports sequential, direct, keyed direct, and indexed access to files. Indexed access requires that the Indexed Access Method program be installed.

Interactive Utility Programs: The manager also supplies interactive utility programs that enable you to:

- Build and modify the screens used in your transaction application
- Create and display the contents of data files that the application uses. These files can be indexed access files or direct access files.
- Disconnect and reconnect terminals to the application
- Send (broadcast) messages to other terminals
- Print a log of transactions or print a screen
- Display reports on terminal, program, and file use
- Assist DEBUG application programs
- Transfer screen displays
- Cold start the manager the next time it is started

Printer Support: The manager supports two kinds of printer:

- Screen printer on which screen images are printed when PF2 is pressed
- Callable PRINT routines that enable buffers to be printed as well as specifying printer attributes including different physical addresses

Terminal Backup: In a multiprocessor system, should one processor fail, its terminals could be restarted by a backup processor. This assumes:

- Two processors connected with a pair of Programmable Two-Channel Switches
- A Multiple Terminal Manager on each processor, each manager servicing its own terminals, but also serving as backup for the other manager's terminals

Managing the Application with the Multiple Terminal Manager

The manager enables multiple transaction programs to share a single partition by swapping your transaction programs to disk, diskette, or secondary storage when the programs are waiting for terminal I/O to be completed. When the terminal I/O for a transaction is completed, the manager swaps the transaction program back into its storage area and continues executing.

The Multiple Terminal Manager can roll out applications when performing system-wide file input/output. This frees up the program area for other applications, thus improving system performance.

Multiple Terminal Support of 3101

The manager supports 3101 terminals in both block and character mode. The IBM 3101 character mode buffers, which occupy 2K bytes each, may optionally be located in secondary storage rather than the Multiple Terminal Manager partition. Additional free space in the partition can be used to run larger user programs.

PASSTHRU Service (3270 Device Emulation)

The manager's PASSTHRU service enables a Series/1 to appear to a host subsystem as one or more remote 3270 Information Display Systems. The manager emulates a 3270 Information Display System (IDS) using the Series/1 as the control unit, any of the manager's full-screen formatted terminals, and a Series/1 printer.

A manager using PASSTHRU communicates with:

- System/370 Time Sharing Option (TSO)
- A host System/370 subsystem (such as IMS/VS or CICS/VS) using one of the following communications protocols:
  - Binary synchronous communications (BSC)
  - Synchronous data link control (SDLC)

PASSTHRU uses the IBM 3270 Information Display also. Version 3 provides PASSTHRU support for IBM 3270 Information Display System devices attached via either BSC Single-Line C (#2074) or BSC 8-Line Control/4-Line Adapter (#2093/2094) utilizing the EBCDIC transmission code.

The following applies to the 3270 Information Display System:

- Formatted full screen support for 1920-character 3270 Display Stations
- 3270 Display contents may be printed on any of the 328X printers. A 328X Printer can be attached to the same control unit as the display; only one 328X Printer per control unit is supported.
- All 3270 control units must be attached to a single BSC multipoint line—only one BSC line with 3270 devices attached is supported per Multiple Terminal Manager task set.

Compatibility with Other Programs

The Query program will operate with the Multiple Terminal Manager.

Documentation

The necessary manual for using the Multiple Terminal Manager is the *Multiple Terminal Manager Version 3 User's Guide.*

Query

The Query licensed program is an aid for transaction applications that process data in indexed or sequential files. Query can be used interactively under the control of the Multiple Terminal Manager or non-interactively via Cobol or assembler language programs.

You use Query to retrieve specific information from your data files. This information may be used, for example, by stock clerks, supervisors, accountants, or managers.

Query consists of:

- A Query processor
- A menu-driven utility
- An application program interface

Query Processor

The Query processor is command-driven. It provides:

- The ability to interactively:
  - Retrieve field information from a sequential file or indexed file. The output of this retrieval can be displayed, printed, or written to a sequential file.
  - Update or change data fields within a record of an indexed file
  - Delete records from an indexed file
  - Insert or add new records to an Indexed Access Method file
- The means to test data fields and select records to be processed using comparison operators such as equal to (=), greater than (>), less than (<), not (/), and their logical combination. Up to ten fields can be qualified to control record selection. Three tests, logically ANDed or ORed, can be specified per field.
- The means to display or print sequenced output records
- The means to store Query definitions together with field test information in a file for later recall and processing (predefined queries)
- A cancel function to stop processing the current request and enable a new request to be started
- The means to produce sorted output files
- Help screens
- Support of user-written security routines
- Optional journaling of all deletions and insertions to indexed files when processing under the Query/Multiple Terminal Manager interface. These maintenance-type requests for records in a file are written to a journal file. This feature provides an audit trail.
- A count function to count the number of records that meet the user-defined field test criteria
- Generic processing for character data fields. Generic processing is performed by specifying some leading character sequence for a field. All fields that contain that leading character sequence are selected. For example; select all names that begin with character(s) S__, SM__, etc.
- Calculation of subtotals and totals for numeric fields
- Global update of a field. This request updates all occurrences of a field in all qualified records in an indexed file, with a single query.

Menu-Driven Utility

The menu-driven utility enables you to:

- Define the fields in a file (file descriptions)
- Edit binary and character numeric fields. (Editing a numeric field means displaying monetary symbols, unit of measure symbols, positioning decimal points and commas, and suppressing leading zeros.)
- Define which fields in a file to process (table descriptions)
- Perform a logical connection of two files. This function supports the need to select all records from two files based on a common field.
- Define how to select and process file information (predefined queries)
- Assign and maintain Query user passwords

Application Program Interface

The application program interface uses a standard CALL interface; it enables programs written in Cobol or Series/1 Assembler to invoke Query functions. If you use Query from a Cobol or assembler language program, Query object modules are linked with the program that runs as a Multiple Terminal Manager program or as a system application program.

Documentation

The necessary manuals for using Query are the *Query User's Guide and Work Book* and the *Query Installation and Programmer's Guide.* General information about the Query program is in the *Query General Information Manual.*

Preparing a Program for Execution

The application builder facility generates a program (task set) that is executable under the Realtime Programming System. The application build step:

- Combines your compiled programs with other subprograms you previously compiled. The application builder INCLUDE statement specifies the programs.
- Provides automatic symbol resolution by incorporating subprograms that your programs invoke. For programs produced using high-level language compilers, these subprograms are those routines that are commonly needed to perform activities such as error handling or storage management. This function is automatic since many programmers do not know that these routines are needed. The application builder AUTOCALL facility performs this function.
- Provides automatic symbol resolution for subprograms and data areas that reside in a shared task set, a shared storage area that several programs can use
- Allocates storage space for certain data areas needed to execute your program. This allocation includes the special processing that is needed to handle Fortran COMMON and PL/I STATIC EXTERNAL data areas.
- Allocates storage for a global data area. A global data area is used to pass data from one program to the next program that executes in the same partition. This facility enables you to divide your application into several task sets that execute in sequence in the same partition, and to pass parameters from one task set to another.
- Puts your program into a format that enables it to be loaded and executed by the operating system. This format is called task set format.
- Assigns the primary entry point to the program. (This is not a common concern for an application written in a high-level language because the compiler assigns the entry point to the program.)
- Optionally preallocates system data areas, called control blocks, that the operating system uses when executing your program. This function is usually necessary for applications that require high performance and must execute as a realtime application.

The application builder is invoked implicitly in the command language facility commands that compile, load, and execute source programs written in PL/I, Cobol, Pascal, Fortran, and assembler language, or can be invoked explicitly using the BLD command.

Documentation

The necessary manual for using the application builder is the *Application Builder User's Guide*. The *Reference Summary* provides summary information.

Handling Large Programs

The application builder provides these facilities to handle very large programs:

- Overlays
- Loadable external modules
- User transients

These facilities enable you to build programs that run when there is insufficient primary storage available.

Overlays

Overlay processing is a way of having subprograms replace (overlay) each other in storage when they are needed.

The operating system supports two kinds of overlays: disk overlays and storage overlays. Both kinds can be used in the same program.

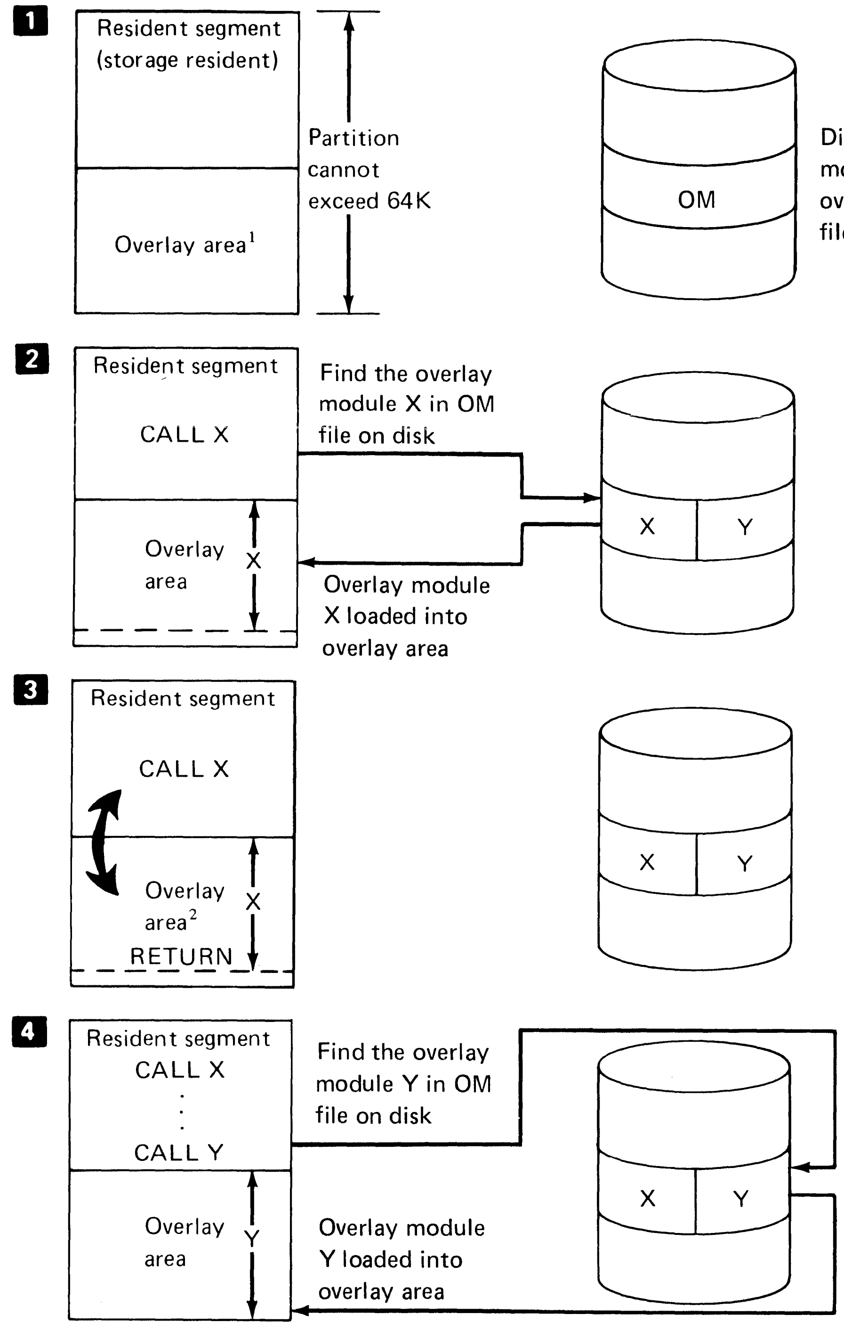
Disk Overlays: **Disk overlays** are segments of programs or entire programs that reside on disk. When a disk overlay is needed by your application, the operating system retrieves the overlay from the disk.

Figure 5-1 on page 5-31 shows four maps of processor storage. These maps show a partition in which an application program is to execute. Assume that the program consists of three major program segments:

- A resident segment or controlling program, with support functions
- A subprogram X
- A subprogram Y

Refer to Figure 5-1 while reading the following:

- In **[1]**, the resident segment is identified to the application builder. This segment resides in storage until execution completes. Execution of the program starts in the resident segment. Note that a partition cannot exceed 64K of primary storage.
- Note the **overlay modules** or segments identified to the application builder. An overlay consists of one or more subprograms. The application builder stores the overlay into a special area in the executable program named OM (for overlay module). The application builder also keeps track of the largest overlay segment it encounters in building the executable program. The size of the largest overlay becomes the size of the overlay area shown in **[1]**.
- In **[2]** the resident segment is loaded, and an overlay program X is called. The application builder has inserted code that retrieves the needed overlay. The overlay is read into the partition's overlay area, and execution starts at the entry point in overlay program X. Within the overlay, you can pass control to other subprograms within the overlay or to subprograms that are part of the resident segment. This process is illustrated in **[3]**.
- When control returns from the overlay module to the resident segment, the overlay area is ready to be reused. This process is shown by the RETURN in **[3]**.
- In **[4]**, subprogram Y is called. The needed overlay module is read from disk and replaces subprogram X in the overlay area. Note that subprogram Y is larger than subprogram X. Remember that the application builder determines the size of the overlay area from the size of the largest overlay.
- In PL/I or Fortran applications, you can make any separately compiled subprograms into an overlay by using the application builder OVERLAY control statement. You do not need to change your code, and you can execute the same program in two different ways by processing it differently in the application builder step.
- Note that one overlay subprogram cannot invoke a subprogram in another overlay. To invoke another overlay, you must return control to the resident segment and have the resident segment invoke the next overlay.
- Cobol applications are overlaid if the segmenting features of the Cobol language are used. These features of Cobol cause the compiler to generate the necessary application builder control statements.


1
Resident segment (storage resident)
Overlay area[1]
Partition cannot exceed 64K
OM
Disk contains overlay modules in an overlay module file named OM
2
Resident segment
CALL X
Overlay area
X
Find the overlay module X in OM file on disk
Overlay module X loaded into overlay area
X
Y
3
Resident segment
CALL X
Overlay area[2]
RETURN
X
X
Y
4
Resident segment
CALL X
CALL Y
Overlay area
Y
Find the overlay module Y in OM file on disk
Overlay module Y loaded into overlay area
X
Y

[1] Allocated by the application builder to be equal in size to the largest overlay in your program

[2] The overlay module can address data and storage areas in the resident segment

Figure 5-1. How Disk Overlays Work

Storage Overlays: **Storage overlays** use unmapped storage to place overlay segments into storage when your application starts execution. As overlays are invoked from the resident segment, the storage overlay window maps the physical storage that contains the needed overlay.

Storage overlays begin executing more quickly than disk overlays because I/O operations to the disk device are unnecessary for each reference to the overlay segment.

Figure 5-2 on page 5-33 shows an overlay structure built by the application builder. The overlay structure contains a window area that is large enough to contain the largest storage overlay. The application builder also informs the operating system how much additional storage is required to hold all the storage overlays in the task set. This area of storage is called the secondary storage pool and can be considered an extension of the partition in which you execute your application. This storage area should not be used for other purposes if it is designated to hold storage overlays.

Refer to Figure 5-2 while reading the following:

- In **[1]** note that the operating system loads all the storage overlay segments into secondary storage and loads the resident segment and storage overlay window into the partition.
- In **[2]** the resident segment invokes subprogram Z which is storage overlay segment 3.

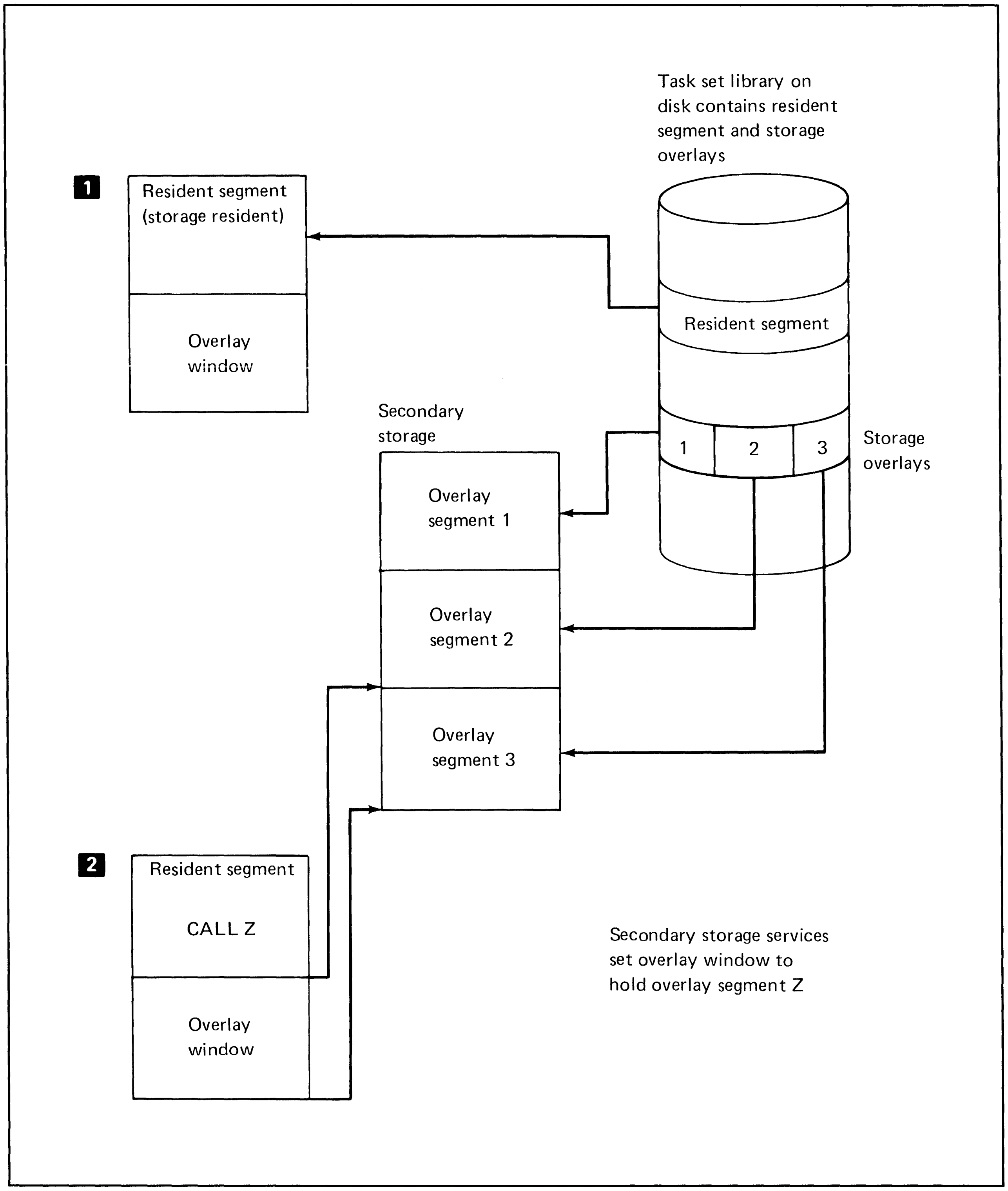

Task set library on disk contains resident segment and storage overlays
1
Resident segment (storage resident)
Overlay window
Resident segment
Secondary storage
1
2
3
Storage overlays
Overlay segment 1
Overlay segment 2
Overlay segment 3
2
Resident segment
CALL Z
Overlay window
Secondary storage services set overlay window to hold overlay segment Z

Figure 5-2. How Storage Overlays Work

Loadable External Modules

Loadable external modules are program modules that are loaded and unloaded anywhere in the storage of your partition or unmapped storage while your task set is executing. Your program loads the module (LOADP macro) when it is needed and unloads it (UNLOADP macro) when it is not. The modules reside in dynamic storage and are created during phase 3 of the application builder. Loadable external modules are available to Cobol, Fortran, PL/I, and assembler language programmers.

Loadable external modules are bound by the application builder to a task set. This task set is the task set that "owns" the module.

When building loadable external modules, the application builder resolves external symbolic references in a loadable external module to symbols in the owning task set or in a shared task set.

Loadable external modules differ from disk and storage overlays in that:

- The owning task set does not have to be bound at once with all possible loadable external modules. Each module can be built independently as needed.
- Because the loadable external modules, in general, are not known to the owning task set when it is built, you reference the loadable external modules with a 1-8 character name
- The modules are loaded anywhere in partition storage and can be relocated at run time

If you design a program to use loadable external modules, the program can be enhanced in the future without adversely affecting your original design. For example, you could design a commercial software package in which each part is a loadable external module. If the original package contained a payroll program and you wanted to add a general ledger program, you could do so. This enables you to expand a single package to meet the needs of many remote sites. If you needed to add more programs later, you could add and integrate them in the package without major rework.

User Transients

User transients are self-relocating programs that run in problem state in a user partition. These programs are not part of the resident portion of the task set, but are loaded by the LOAD macro from disk into dynamic storage in a user partition. User transients enable you to have multiple programs share space at different times during the execution of a program. For more information on user transients, see the *Supervisor Services Programming Guide.*

Additional Application Tools

Additional application tools include the extended operating system support to handle indexed files and to sort and merge data.

Indexed Access Method

The Indexed Access Method licensed program provides access to data by key. This support is an extension to the operating system input/output routines. The Indexed Access Method supports both batch and multiuser interactive applications.

The data file organization for indexed files enables both direct and sequential processing. A cascading index technique is used for direct access. Sequence chaining of data blocks is used for sequential processing.

The access method design supports files that have high add/delete activity (such as open order files) to minimize performance degradation. High performance is achieved by distributing free space for additions throughout the file, by updating and inserting additions in place, and by dynamically reclaiming space after deletions.

Performance of application programs is improved by the asynchronous processing capability of the Indexed Access Method. Under control of your application program, I/O requests to the Indexed Access Method cause a return to your program before the I/O operation is completed. Thus, computation can overlap with I/O operations.

The Indexed Access Method enables multiple tasks and multiple programs (task sets) to share the same files. Data integrity is maintained by:

- System-wide or node-wide (in a multiprocessor system) record-level, block-level, and file-level locking to prevent multiple concurrent updates of the same record
- Immediate write-back of updated records
- An exclusive option, which specifies a file is for the exclusive use of the requester

Only fixed-format data records are supported. Keys must be unique for each index and are fixed in length (up to a 254-character maximum). Keys are imbedded and contiguous in the data record. You can use more than one index to access a file by assigning a different contiguous area as an alternate key. Each index accesses the file by its own key.

The Indexed Access Method program:

- Provides an interactive utility (IUTIL) with commands for defining and creating an indexed file. Using the utility, a high-level language programmer need never use assembler language to create an indexed file.
- Provides data paging so recently used blocks are retained in processor storage for rapid access
- Opens and closes a file. Indexed files are closed and resources are freed when a program using an indexed file terminates either normally or abnormally.
- Provides error log information
- Utilizes a dynamic file structure capability so file structure adjusts itself to handle record additions and deletions
- Provides a utility (VERIFY) for checking the integrity of the index structure and print control blocks, and for printing a free space report for an indexed file

- Writes a record of data with a key. Output operations include:
  - Replacing (updating) a previously read record and verifying that the key field has not been modified
  - Inserting a new record with a unique key
  - Deleting a record
  - Freeing a record lock to avoid replacing an unmodified record
- Reads a record of data by key. Input operations include:
  - Retrieving a record (read only)
  - Retrieving a record with a lock (in preparation for updating the record)
  - Retrieving a record whose key is equal to, greater than, or greater than or equal to the requested key
  - Retrieving the next record (in ascending key order) from the point of last access
- Enables you to create a free area to contain reserved blocks of space. These blocks are later allocated as data blocks or index blocks to expand a portion of a file with heavy insert activity in that part of the file.
- Enables you to extract information about a file including key length, key displacement, block size, and record size
- Supports a new SVC number, which the macros generate when used in an assembler language program. The old SVC number, 192, will continue to be supported at execution time at the option of the user.
- Can be started and stopped via operator commands. Thus, it is not present in main storage until it is started.

Used in a multiprocessor environment, the Indexed Access Method program can:

- Be started on each node with indexed files so that it can process those files
- Provide a backup Indexed Access Method which will assume control of access to user files at another node under certain failure conditions
- Process Indexed Access Method function requests from application programs on any node. Each request is automatically routed for processing to the node where the file resides.

Cobol and PL/I programmers can access indexed files from an application program. The Sort/Merge program accepts an indexed file as input. The Multiple Terminal Manager can also access indexed files using Multiple Terminal Manager file I/O services.

Indexed Access Method in a Multiprocessor System

In a multiprocessor system, the Indexed Access Method program can be copied and located at each node having indexed files. Indexed Access Method programs on different nodes can be paired so they provide backup capabilities for one another: if one of the paired programs fails, the other will take over its workload in addition to its own. Functions of the Indexed Access Method program can be requested from application programs on any node. Each request is routed to the proper node for processing. Responses are routed back to the requester.

Loadable Indexed Access Method enables nodes with and without the Indexed Access Method to be installed on the standard system. Installation and utility functions enable the Indexed Access Method control data sets to be duplexed at the option of the user.

Indexed Access Method Pool

You can improve the performance of your applications that use indexed files by enabline the Indexed Access Method to allocate a data-paging pool to hold index and data pages. The Indexed Access Method optionally allocates a number of 2K-byte pages for use as a pool. The storage for this pool is allocated from secondary storage.

As Indexed Access Method data and index blocks are accessed, they are retained in this storage pool. Thus, they can be accessed quickly if they are requested again. When the pool fills up, the Indexed Access Method replaces the least recently used pages with data it is currently accessing. In this manner, the most frequently accessed index or data blocks are retained.

Documentation

The necessary manual for using the Indexed Access Method is the *Indexed Access Method User's Guide*.

Sort/Merge

The IBM Series/1 Sort/Merge program sorts and merges records from up to eight input files into one output file. Records are sorted in either ascending or descending order. You specify one or more control fields in the records to be sorted. The Sort/Merge program then compares the control fields to determine the relative sequence of the records. Files to be sorted or merged can have any file organization that the operating system supports.

Sort/Merge can be executed interactively at a terminal, as a batch job, or from within a PL/I, Cobol, or assembler language source program.

Series/1 Sort/Merge:

- Accepts multiple volume diskettes in basic exchange format for input or output
- Accepts variable or fixed format records in unblocked, blocked, or spanned form from a system-formatted disk or diskette
- Enables characters to be inserted in sorted records. These characters either control the sort or are to appear in the sorted output as editing characters, or both.
- Permits user exit routines to handle I/O errors
- Enables you to specify either the EBCDIC or ASCII collating sequences
- Provides statistics on the number of records that are processed at different points in the program code or on error conditions arising during execution
- Enables you to invoke multiple sorts from the same application
- Enables you to route messages to the operator console or to a printer
- Selects all or some of the input records by recognizing:
  - Record code
  - Relation of field to a constant
  - Relation of two fields within a record
  - Any relationship in a series (OR'ing)
  - All relationships in a series (AND'ing)
  - Multiples of the above conditions in any combination

Output from the Sort/Merge program is one of four types:

- Address sort
- Record sort
- Record summary sort
- A merge

An address sort supports one input file and produces an output file containing 32-bit binary numbers that are used as an index to the input file. These relative record displacements reflect the sequence specified for the sort job and enable processing of the input file in an ordered manner after the sort job.

The output from a record sort is a file of sorted records in ascending or descending order. You determine the format and content of each sorted output record.

The output from a record summary sort is a file of sorted records that can contain summary records. The summary record accumulates totals from one or more fields in the input records and provides a summary record for each set of input records having control fields with the same value.

A merge operation combines up to eight previously sorted files into one sorted consecutive output file.

Documentation

To use the Sort/Merge program you need the *Sort/Merge Programmer's Guide.*

Summary

Program-development language compilers and subroutine libraries that are available with the Realtime Programming System include:

- Cobol compiler and resident library
- Cobol transient library
- Fortran compiler and object support library
- Mathematical and functional subroutine library
- Fortran realtime subroutine library
- Pascal compiler
- PL/I compiler and resident library
- PL/I transient library
- Macro assembler with structured programming macros

To prepare programs for the Series/1 Realtime Programming System on a System/370, the System/370 Host Preparation Facilities Programming RPQ includes the:

- System/370 Host Assembler
- System/370 Host Application Builder

Interactive tools that support development or production systems include:

- Command language facility (including the S1/EXEC interpreter commands)
- Text editor
- System utilities
- Tape utilities
- Stand-alone utilities

The following programs support non-interactive batch processing under the Realtime Programming System:

- The job stream processor component of the Program Preparation Subsystem (for development environments)
- The Job Stream Processor Programming RPQ (for production environments)

The following program constructs a program (task set) that is capable of executing under the Realtime Programming System:

- The application builder component of the Program Preparation Subsystem

The following programs provide additional application tools:

- Multiple Terminal Manager
- Query
- Indexed Access Method
- Sort/Merge

Index

IBM Series/1 Realtime Programming System Version 6:
Concepts and Facilities

Order No. GC34-0471-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

GC34-0471-1
Printed in U.S.A.

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Information Development, Department 28B
P.O. Box 1328
Boca Raton, Florida 33432

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

IBM Series/1 Realtime Programming System Version 6:
Concepts and Facilities

Order No. GC34-0471-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

GC34-0471-1
Printed in U.S.A.

Reader's Comment Form

Cut or Fold Along Line

Fold and tape   Please Do Not Staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS   PERMIT NO. 40   ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Information Development, Department 28B
P.O. Box 1328
Boca Raton, Florida 33432

Fold and tape   Please Do Not Staple   Fold and tape

IBM®

International Business Machines Corporation

GC34-0471-1
Program Number 5719-PC6
File No. S1-20
Printed in U.S.A.