SERIES 60 (LEVEL 6)

GCOS 6 MOD 400 SYSTEM CONCEPTS ADDENDUM A

SUBJECT

Changes and Additions to the Manual

SPECIAL INSTRUCTIONS

Insert attached pages into Revision 0 of the manual dated January 1978 (see Collating Instructions). Except for Appendix C which has been completely revised, change bars indicate new and changed information; asterisks denote deletions.

Note:

Insert this addendum cover behind the manual cover to indicate that the manual is updated with Addendum A.

SOFTWARE SUPPORTED

This update supports Release 0110 of the Series 60 (Level 6) GCOS 6 MOD 400 Software System. For any later release of MOD 400 software, see the Manual Directory of the latest *System Concepts* manual to ascertain whether this manual supports that release.

ORDER NUMBER

CB20A, Rev. 0

June 1978

20997
3.5678
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove	Insert
iii, blank	iii, iv
v, vi	v, vi
vii, blank	vii, blank
1-1, 1-2	1-1, blank
—	1-1.1, 1-2
1-3, 1-4	1-3, blank
	1-3.1, 1-4
1-5, 1-6	1-5, 1-6
	1-6.1, blank
1-7, 1-8	1-7, 1-8
1-9, blank	1-9, blank
Section 2	Section 2
3-5 through 3-8	3-5 through 3-8
—	3-8.1, blank
3-9, 3-10	3-9, 3-10
3-11, blank	3-11, blank
4-1, 4-2	4-1, 4-2
—	4-2.1, blank
4-3, 4-4	4-3, 4-4
5-1, 5-2	5-1, 5-2
5-5 through 5-10	5-5 through 5-10
—	5-10.1, 5-10.2
5-11, 5-12	5-11, 5-12
6-5, 6-6	6-5, 6-6
—	6-6.1, 6-6.2
Section 7	Section 7
Appendix C	Appendix C
D-3, D-4	D-3, D-4

File No.: 1S13

6/78
CB20A

MANUAL DIRECTORY

The following publications constitute the GCOS 6 Mod 400 manual set. They support Release 0110 of GCOS 6 Mod 400 and fully describe its major components.

Order No.	*Manual Title*	*Revision*	*Date*
CB01	*GCOS 6 Program Preparation*	1	June 1978
CB02	*GCOS 6 Commands*	1	June 1978
CB03	*GCOS 6 Communications*	1	June 1978
CB04	*GCOS 6 Sort/Merge*	0	January 1978
CB04A	Addendum A		June 1978
CB05	*GCOS 6 Data File Organizations and Formats*	0	January 1978
CB05A	Addendum A		June 1978
CB06	*GCOS 6 System Messages*	0	January 1978
CB06A	Addendum A		June 1978
CB07	*GCOS 6 Assembly Language Reference*	1	June 1978
CB08	*GCOS 6 System Service Macro Calls*	1	June 1978
CB09	*GCOS 6 RPG Reference*	0	January 1978
CB09A	Addendum A		June 1978
CB10	*GCOS 6 Intermediate COBOL Reference*	1	June 1978
CB20	*GCOS 6 Mod 400 System Concepts*	0	January 1978
CB20A	Addendum A		June 1978
CB21	*GCOS 6 Mod 400 Program Execution and Checkout*	0	January 1978
CB21A	Addendum A		June 1978
CB22	*GCOS 6 Mod 400 Programmer's Guide*	0	January 1978
CB22A	Addendum A		June 1978
CB23	*GCOS 6 Mod 400 System Building*	1	June 1978
CB24	*GCOS 6 Mod 400 Operator's Guide*	0	January 1978
CB24A	Addendum A		June 1978
CB25	*GCOS 6 Mod 400 FORTRAN Reference*	0	January 1978
CB26	*GCOS 6 Mod 400 Entry-Level COBOL Reference*	0	January 1978
CB27	*GCOS 6 Mod 400 Programmer's Pocket Guide*	1	July 1978
CB28	*GCOS 6 Mod 400 Master Index*	0	April 1978
CB30	*Remote Batch Facility User's Guide*	0	January 1978
CB30A	Addendum A		June 1978
CB31	*Data Entry Facility User's Guide*	1	June 1978
CB32	*Data Entry Facility Operator's Quick Reference Guide*	0	June 1978
CB33	*Level 6/Level 6 File Transmission Facility User's Guide*	1	June 1978
CB34	*Level 6/Level 62 File Transmission Facility User's Guide*	0	January 1978
CB34A	Addendum A		February 1978
CB34B	Addendum B		June 1978
CB35	*Level 6/Level 64 (Native) File Transmission Facility User's Guide*	0	January 1978
CB35A	Addendum A		June 1978
CB36	*Level 6/Level 66 File Transmission Facility User's Guide*	1	June 1978

CB37	*Level 6/Series 200/2000 File Transmission Facility User's Guide*	0	January 1978
CB37A	Addendum A		June 1978
CB38	*Level 6/BSC2780/3780 File Transmission Facility User's Guide*	1	June 1978
CB39	*Level 6/Level 64 (Emulator) File Transmission Facility User's Guide*	0	January 1978
CB39A	Addendum A		June 1978
CB40	*IBM 2780/3780 Workstation Facility User's Guide*	0	June 1978
CB41	*HASP Workstation Facility User's Guide*	0	June 1978
CB42	*Level 66 Host Resident Facility User's Guide*	0	June 1978
CB43	*Terminal Concentration Facility User's Guide*	0	June 1978

In addition to the GCOS 6 Mod 400 manual set, the following publications are available.

Order No.	*Manual Title*
AS22	*Honeywell Level 6 Minicomputer Handbook*
AT04	*Level 6 System and Peripherals Operation Manual*
AT97	*MLCP Programmer's Reference Manual*
FQ41	*Writable Control Store User's Guide*
CC54	*INFO 6 User's Guide*
CC62	*INFO 6 Quick Reference Pocket Guide*
CB77	*TOTAL Data Base Management System Reference Manual*
CC77	*Level 6 Technical Notes*

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 1
SYSTEM CHARACTERISTICS

GCOS 6 MOD 400 software is a disk-based operating system that supports multitasking, real-time, or data communications applications in one or more online streams. In addition, program development or other batch type applications can be performed concurrently in a single batch stream.

GCOS is a multifunctional system, capable of supporting a variety of processing functions. The user can develop and execute applications software, perform forms data entry, transmit files to other computers, and enter jobs for execution at remote sites. Terminal concentration services and IBM workstation facilities are also available.[1]

The system can be configured to process different functional applications concurrently. For example, a user can run his own applications, utilize other system functionality such as the data collection capability, and communicate with a host processor, all at the same time.

SOFTWARE FEATURES

The operating system includes Monitor, File System, and data communications facilities as well as an extensive set of program preparation components, utility routines, and debugging aids. Additionally, the operating system supports various software packages for implementing a distributed processing environment. Also available are software routines to accommodate applications developed under the GCOS/BES (Basic Executive System) Operating System.

The Monitor supports the execution of user application tasks and provides a set of system services that enables users to control execution of individual tasks and to synchronize multiple tasks with one another and with time-related events. The Monitor controls the loading of user programs and manages requests for available memory. It provides standard system trap handling routines for responding to exception conditions and also allows users to provide their own trap handling routines for user-caused trap conditions.

The File System software offers an extensive set of logical I/O access methods. It provides device-independent access to any device for sequential files, and direct access to disk files. In addition the File System software automatically manages the space utilization of mounted disk volumes, thus allowing users to create, expand, and release disk files as required by online applications needs.

The operating system offers two levels of communications interface. Remote/local terminals may be accessed through the sequential file interface of file management, or for more direct control of the communications environment, by using the systems physical I/O interface. The communications facility includes line protocol handlers for teleprinter and VIP devices, binary synchronous communications (BSC), and the polled VIP emulator (PVE).

The software includes a powerful and comprehensive set of program preparation components, utilities and debugging aids for applications developments, all running under control of the Monitor. Programming languages include assembly language, RPG, FORTRAN, and Entry-Level and Intermediate COBOL. A Commercial Central Processor Model and a Scientific Instruction Processor (SIP) or equivalent software simulators are available with the system. The RPG and Intermediate COBOL Compilers generate code for the CIP: the FORTRAN Compiler generates SIP code. The Assembler supports both CIP and SIP instructions.

[1] A host resident facility is available on Series 60 Level 66 central systems; the host resident facility permits use of Level 66 resources to develop applications for the Level 6.

The system supports various software packages for implementing functional links to other, remote processors. These components interface with the communications software to enable use of the Level 6 in a true distributed processing environment. The various software packages are described below.

- The file transmission capability supports transmission of files between the Level 6 and Series 60 (Level 6, 62, 64, or 66) or Series 2000 processors, using the polled VIP protocol; or between the Level 6 and non-Honeywell processors, using the BSC protocol.
- The Remote Batch Facility (RBF) permits a Level 6 system to be used for job submission and output delivery for one or more Series 60 (Level 66) or Series 6000 host processors, using the Remote Computer Interface (RCI) or High-Level Data Link (HDLC) protocols. Local processing (such as program development and user application execution) can occur concurrently with remote batch processing.
- The Data Entry Facility (DEF) provides a data collection capability that includes creation/modification of forms; formatted data input; validation, extraction and verification of data; and formatted printing of data. The facility supports multiple independent operator display stations that use VIP devices.
- A Terminal Concentration Facility permits various types of synchronous and asynchronous terminals to concurrently connect to a Level 6 and have their message traffic concentrated (multiplexed) over one or more links to a host processor. The Terminal Concentration Facility smooths sporadic terminal traffic patterns, reduces the number of modems and cross-country lines required by multi-terminal applications, and improves total reliability.

- IBM workstation facility software includes the IBM 2780/3780 Workstation Facility and the HASP Workstation Facility. The facilities provide a means of communicating with an IBM 360/370 host system.

GCOS 6 MOD 400 facilities accommodate applications developed under the GCOS/Basic Executive Systems (BES). These facilities include utilities to move source and object program files between the two operating systems, and support of BES system service calls via a special * interface package. Files created under the BES File Manager are supported directly by MOD 400; programs created under BES must be relinked under MOD 400.

OPERATING FEATURES

The operating system supports concurrent execution of multiple tasks running under one or more task groups. Each task group owns the resources necessary for execution of an application program (one or more related tasks). The task group runs independently in its own operating environment while sharing the resources of the operating system.

Multiprogramming can be achieved by defining more than one application task group to be run concurrently. Serial execution of tasks in a task group can be accomplished by stepping through execution of the tasks in sequence; multitasking can be achieved by causing tasks in the group to be executed concurrently.

Multiple online task groups can be run concurrently with a single batch task group. The batch task group (used for program development or a batch-oriented user application) can be rolled out to a disk to obtain additional memory for online applications.

The number of task groups being run is limited only by the amount of memory available. Concurrently executing task groups may occupy independent, dedicated memory areas, or they may contend for space within a memory pool. When one task group is deleted, the released memory is available to other task groups in the same pool. The Monitor allocates memory dynamically from pools and can relocate programs at load time. Once a task group requests execution, it is dispatched according to its assigned priority level. When multiple tasks share a priority level, they are serviced in a round-robin fashion.

Use of the file system by multiple independent users is facilitated by the arrangement of file system entries (directories and files) in a tree-structured hierarchy. Each directory or file is identified by a pathname that indicates the path from the root directory of the hierarchical structure to the particular directory or file. File reference is simplified through the use of pathnames relative to a working directory that indicates a user's current position in the file system hierarchy. Access to sharable files and devices is controlled by file attributes and concurrency procedures.

SUMMARY OF SYSTEM FEATURES

The GCOS 6 software offers the following capabilities:

- Provides a multi-user operating system
- Supports multiple concurrent programming environments, with applications being run in one batch stream and one or more online streams
- Controls program preparation through the operating system
- Handles program preparation and execution of user applications concurrently
- Handles execution of multiple user applications
- Permits multitask execution within each user application
- Controls the execution sequence of user applications
- Supports real-time operations
- Provides communications support
- Is time and interrupt driven
- Allows device independent programming
- Supports program overlay capability

- o Provides four programming languages: assembly language, FORTRAN, RPG, and two levels of COBOL (entry level and intermediate)
- o Provides a hierarchical file system with extensive utility support
- o Supports four disk file organizations
- o Supports code sharing via reentrant programs
- o Permits multiple functions that interface with communications facilities to be run concurrently with application development and execution
- o Supports file transmission between the Level 6 and other computers
- o Provides the Remote Batch Facility, permitting the Level 6 system to be used for job submission to a host processor
- o Provides the Data Entry Facility, permitting forms creation and data collection
- o Provides compatibility software for GCOS/BES and GCOS/BES2 programs and files
- o Provides a Terminal Concentration Facility, permitting the concentration (multiplexing) of message traffic.
- o Provides IBM workstation facilities, permitting communication between the Level 6 and an IBM 360/370 system.
- o Supports data sharing via the record locking facility.
- o Supports file protection via access control commands.

GUIDE TO USING THE MANUAL SET

A guide to the use of the manual set is provided below. Information is tailored for specific classes of users applications programmers, systems programmers, and operators. (As used in this guide, the applications programmer writes applications programs; the system programmer configures the system and defines the environment for each application; the operator operates the system from the operator terminal.) Included as a separate subsection is a guide for those who will use the Level 6 in a distributed processing environment.

Applications Programmer's Manual Guide

Figure 1-1 illustrates the suggested sequence for using the manuals. To familiarize yourself with the system facilities, use the *System Concepts* manual. To write an application program, begin by using the *Programmer's Guide* manual. It illustrates: (1) various ways to gain access to the system, (2) a sample Editor session, and (3) for application languages, the procedure for performing program preparation and execution. Working with the small subset of system commands used within examples is a good approach to learning the system command set. This approach for getting started assumes that a system programmer has already configured and started up a suitable application environment.

Through examples, the *Programmer's Guide* illustrates how to use the system facilities. Other manuals provide reference material. The *Program Preparation* manual contains Editor directives (statements) to create and update an application language source unit. For each of the languages, the appropriate language reference manual contains the description of the language statements. Operating system dependencies, if any, that affect how you write the application are described in the *Programmer's Guide*. If the application uses communications, refer to the *Communications Processing* manual. Read the *Data File Organizations and Formats* manual if you require a better understanding of a language-supported file organization that is to be used in an application or if you must calculate the size of a data file. You can use Monitor macro calls, as described in the *System Service Macro Calls* manual, in assembly language programs. Before your program can be entered for execution, it must be linked as described in the *Program Execution and Checkout* manual.

For program compilation or assembly and execution, the procedures described in the *Programmer's Guide* might be sufficient. To obtain more control over the execution of your program or utilize the system facilities more completely or efficiently, use the commands described in the *Commands* manual. If you wish to use the operator terminal, read the *Operator's Guide* to learn how to use that terminal. In many cases, the description of commands must be supplemented by system concepts described in the *System Concepts* manual. Rather than read all the conceptual material at one time, you may find it more

meaningful to refer to it in conjunction with the appropriate reference material. The *Commands* manual also describes the utilities. The Patch, Debug, and Dump utilities are described in the *Program Execution and Checkout* manual; file transmission from Level 6 to a host system is described in the appropriate *File Transmission* manual. Error messages and return status codes are listed in the *System Messages* manual.

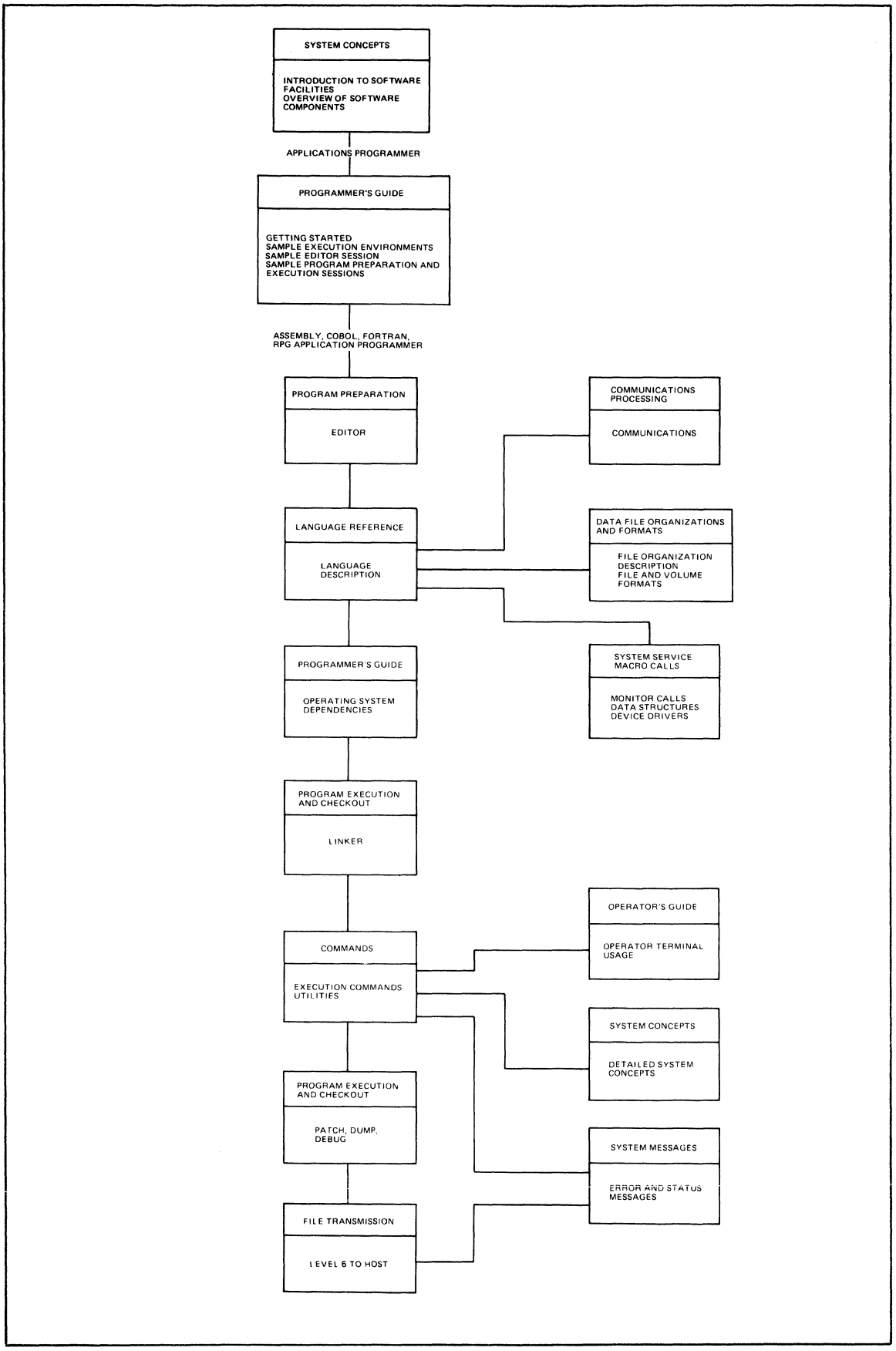

Figure 1-1. Applications Programmer's Guide to Manuals

System Programmer's Manual Guide

Figure 1-2 illustrates the suggested sequence for using the manuals. The *System Building* manual provides you with the configuration directives (statements) and startup procedures to configure and start up a MOD 400, a Remote Batch Facility (RBF), or a Data Entry Facility (DEF) system. You must know the conceptual material in the *System Concepts* manual in order to successfully use the configuration directives. To tailor an applications environment suitable for the intended application, you use operator commands described in the *Operator's Guide* manual. Error messages are listed in the *System Messages* manual. If you are working with an application that runs under the BES operating system, the *System Concepts* manual contains MOD 400 and BES compatibility considerations.

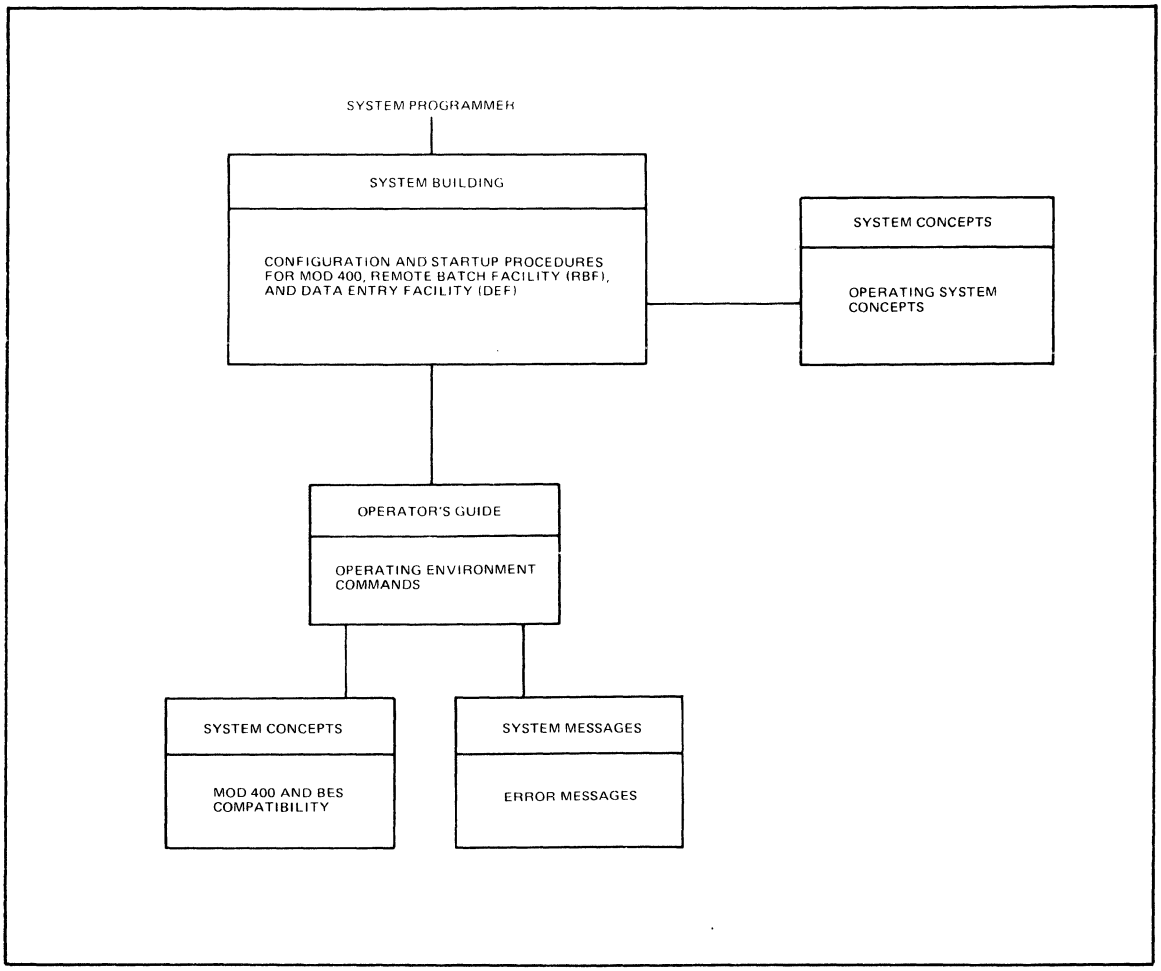

Figure 1-2. System Programmer's Guide to Manuals

Operator's Manual Guide

Figure 1-3 illustrates the suggested sequence for using the manuals. Specific operator job functions must be determined by each installation; a large system might have a person assigned as an operator; a small system might have each programmer also act as an operator. The *Operator's Guide* indicates those system procedures performed through the operator terminal and describes operator commands used in system operation.

The *Programmer's Guide* contains examples using commands (described in the *Commands* manual) that are similar to operator commands. The *System Concepts* manual provides an understanding of the operating system. Note that the *Operator's Guide* describes using the operator terminal for operator functions to enter operator commands to the system task group, or for user functions to enter commands to a user task group. To run the utilities, use the commands (described in the *Commands* manual) entered through the operator terminal functioning as a user terminal. Error messages are listed in the *System Message* manual.

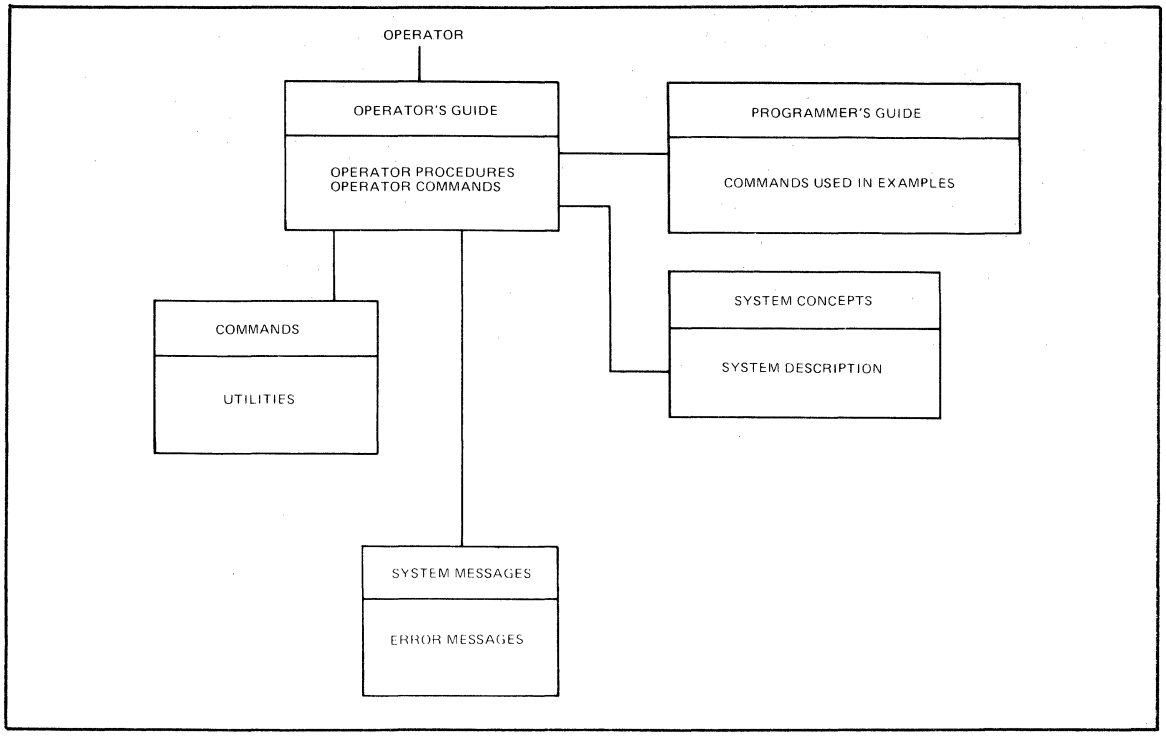

Figure 1-3. Operator's Guide to Manuals

Guide for Using the Manuals in a Distributed Processing Environment

GCOS 6 Mod 400 supports the use of Level 6 in a distributed processing environment. Using Honeywell-supplied software packages, processing capability can be assigned to sites remote to the host computer system. With the functional links provided by Honeywell, a Level 6 can be configured as a host processor and specialized processing (e.g., forms data entry) assigned to remote terminals. Also, the user can develop links with non-Level 6 host processors and distribute the total processing load between the host and Level 6.

The software packages available to the user include the Data Entry Facility, Remote Batch Facility, Terminal Concentration Facility, File Transmission Facility, Host Resident Facility (Level 66), and IBM workstation facility software. Figure 1-4 indicates the documentation available for the operation and use of such software. Configuration information, if applicable, is contained in the *System Building* manual. Commands and error messages are contained in the *Commands* and *System Messages* manuals respectively.

SOFTWARE DOCUMENT SET

This *System Concepts* manual briefly describes GCOS software, system features, and operating concepts. Most of the background information needed to use the reference material in other manuals of this set is presented in Section 3 through 6 of this manual. Except

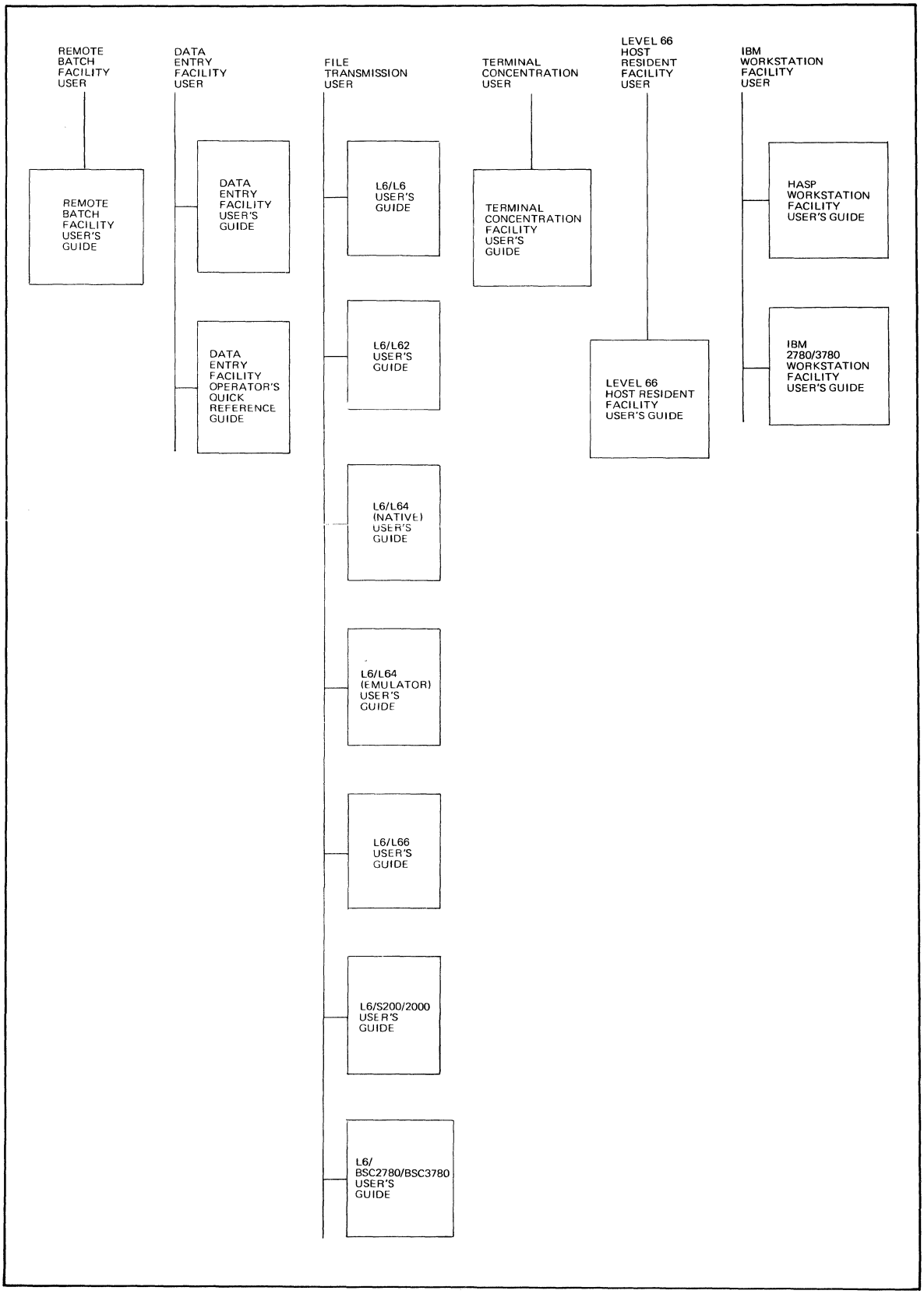

Figure 1-4. Guide for Using the Manuals in a Distributed Processing Environment

for summaries, this material is not duplicated in other manuals and covers the following subjects:

- o Task groups and tasking
- o Memory definition and use
- o Operating system features
- o File system and communications concepts
- o Operating environment configuration

This manual is the sole source for reference material on the compatibility of GCOS/BES1 and BES2 programs and files with a MOD 400 system. Programming conventions are presented in Appendix B of this manual, and a glossary of GCOS 6 MOD 400 terms is in Appendix D.

The contents of other documents in the manual set are summarized briefly below.

- o *GCOS 6 Program Preparation,* Order No. CB01 – Overview of the programming steps to prepare a program for execution. Suffix conventions for files used in program preparation. Detailed description of Editor. Rules for writing assembly language programs using SLIC (SAF/LAF independent code).
- o *GCOS 6 Commands,* Order No. CB02 – Description of command line format, task interrupt break function, activating an application program, and extending the command set. Detailed description of commands, utilities, and language processor execution. Description of additional command line arguments, terminal characteristics at login, Intersystem Link (ISL) directives, and File Change directives; ASCII and EBCDIC character sets.
- o *GCOS 6 Communications Processing,* Order No. CB03 – Introduction to communications software. Description of communications processing through COBOL, assembly language, File System and FORTRAN; sample communications programs; Dump MLCP (DUMCP) utility; TTY, VIP, and BSC control characters; ASCII and EBCDIC character sets.
- o *GCOS 6 Sort/Merge*, Order No. CB04 – Description of the Sort and Merge program features, statement formats, and report contents. Includes file and memory requirements, operating procedures, sample programs, using Sort as a subroutine, debug mode execution, and ASCII collating sequence.
- o *GCOS 6 Data File Organizations and Formats*, Order No. CB05 – Description of disk and magnetic tape data file organizations support for application programs; disk and magnetic tape record, file, and volume formats; unit record file formats; file and volume headers; ASCII and EBCDIC character sets.
- o *GCOS 6 System Messages*, Order No. CB06 – Description of messages reported by system components. Procedure for adding user messages.
- o *GCOS 6 Assembly Language Reference*, Order No. CB07 – Complete description of all instructions, instruction formats, control statements, types of data handled, and macro language statements. Description of Scientific Instructions and Commercial Instructions.
- o *GCOS 6 System Service Macro Calls*, Order No. CB08 – Description of macro call syntax, register and addressing conventions. Detailed description of system services macro calls for the Monitor and File System and for defining data structures; physical I/O device drivers; Trap Handler; Monitor and File System data structures; writing a user device driver; contents of registers for system service macro calls; ASCII and EBCDIC character set.
- o *GCOS 6 RPG Reference*, Order No. CB09 – Complete description of RPG data processing including: a primer on RPG programming, RPG specification form entries, description and use of the RPG fixed logic cycle, and operating instructions with sample programs.
- o *GCOS 6 Intermediate COBOL Reference*, Order No. CB10 – Complete description of the general features of Intermediate COBOL programs, language elements, language syntax, the four major divisions of an Intermediate COBOL program, specific format

descriptions of all Intermediate COBOL statements (including programming examples incorporating each statement), and the types of files and data handled, ASCII collating sequence, COBOL glossary, comparison of standard COBOL with Intermediate COBOL, and Intermediate COBOL run-time considerations.

- *GCOS 6 MOD 400 Program Execution and Checkout*, Order No. CB21 – Overview of program execution sequence. Detailed descriptions of Linker, Debug, Patch, Dump Memory (MDUMP), Dump Edit (DPEDIT), and interpreting and using memory dumps. Table of system service macro calls ordered by function code.
- *GCOS 6 MOD 400 Programmer's Guide*, Order No. CB22 – Description of various possible programming environments at an installation and the ways to access the system for each environment. Sample Editor session. Examples illustrating how to prepare and execute COBOL, FORTRAN, SORT and assembly language programs; how to call FORTRAN routines from an Entry-Level COBOL main program; and FORTRAN chaining. Explanation of headers on listings.
- *GCOS 6 MOD 400 System Building,* Order No. CB23 – Description of system configuration and startup procedures for the MOD 400 operating system; the software packages for distributed processing (as necessary); configuration directives; system disk layout; system overlays; minimum system hardware and configuration requirements to do program preparation; and startup halts. Description of procedures to transfer files to system disk; create a single-diskette system; place a shared version of a utility in the system library; and load and execute the Intersystem Link (ISL) loader of ISL configuration.
- *GCOS 6 MOD 400 Operator's Guide*, Order No. CB24 – Description of routine system startup, activation of the login capability, system operator interface with the system (OIM), operator commands; task interrupt break function from the operator terminal; additional operator command line arguments; listener component setup for login capability; system halt conditions; ASCII character set.
- *GCOS 6 MOD 400 FORTRAN Reference*, Order No. CB25 – Complete description of all statements, instruction formats, types of files and data handled, FORTRAN run-time support routines (intrinsic functions, tasking, I/O), and compiler diagnostics.
- *GCOS 6 MOD 400 Entry-Level COBOL Reference*, Order No. CB26 – Complete description of the general features of Entry-Level COBOL programs, language elements, language syntax, the four major divisions of an Entry-Level COBOL program, specific format descriptions of all Entry-Level COBOL statements (including programming examples incorporating each statement), and the types of files and data handled, compiler diagnostics, ASCII collating sequence, COBOL glossary, comparison of standard COBOL with Entry-Level COBOL, and Entry-Level COBOL run-time considerations.
- *GCOS 6 MOD 400 Programmer's Pocket Guide,* Order No. CB27 – A pocket-size summary of commonly-used commands, directives, and operating procedures as well as a brief description of each coded error message.
- *GCOS 6 MOD 400 Master Index,* Order No. CB28 – An index of specific topics related to GCOS 6 MOD 400. Topics are listed alphabetically. Each topic is keyed to the order number of each manual in which the topic is described.
- *Remote Batch Facility User's Guide*, Order No. CB30 – Description of remote batch operations: communicating with the host, preparing job decks, managing job streams, input and output processing, operator commands and messages, host software control records.
- *Data Entry Facility User's Guide*, Order No. CB31 – Operation of the Data Entry Facility. Description of operation of the Operator Display Station; forms and table development; data entry and verification process; file printing; system supervisory and utility operations; interfacing with data entry and applications programs; and error and system messages.
- *Data Entry Facility Operator's Quick Reference Guide,* Order No. CB32 – A quick reference guide to procedures for operator data entry, data verification, data modification and file printing for the Data Entry Facility.

- *Level 6/Level 6 File Transmission*, Order No. CB33
- *Level 6/Level 62 File Transmission*, Order No. CB34
- *Level 6/Level 64 (Release 0300) File Transmission*, Order No. CB35
- *Level 6/Level 66 File Transmission*, Order No. CB36
- *Level 6/Series 200/2000 File Transmission*, Order No. CB37
- *Level 6/BSC2780/3780 File Transmission,* Order No. CB38
- *Level 6/Level 64 (Release 0220) File Transmission*, Order No. CB39

  Each of the above documents describes the capability of the particular file transmission facility, including file organizations supported, code sets, line protocols, and equipment requirements. Individual sections of the manuals provide the operating information necessary to perform a file transfer from either end of a network (Level 6 and host).
- *IBM 2780/3780 Workstation Facility User's Guide,* Order No. CB40
  HASP Workstation Facility User's Guide, Order No. CB41

  Each of the above documents describes the capabilities of the particular workstation emulation facility, including an overview of the facility, the commands and parameter strings to be entered at the workstation, and programming information required to interface with the IBM host system.
- *Level 66 Host Resident Facility User's Guide,* Order No. CB42 – A description of those Level 66 software facilities that permit use of Level 66 resources to develop applications for the Level 6.
- *Terminal Concentration Facility User's Guide,* Order No. CB43 – Description of Terminal Concentrator Operations including startup and shutdown and a description of terminal operations using the concentrator facility.
- *Honeywell Level 6 Minicomputer Handbook,* Order No. AS22 – Descriptions of hardware models, central processor, processor architecture and features, instruction set, registers, peripheral devices, control panel, software, various controllers and system features, as well as maintenance and site preparation information.
- *Level 6 System and Peripherals Operation,* Order No. AT04 – Description of the operation of the central processor control panels, the operation of each peripheral attachable to Level 6 hardware, and operator trouble-shooting procedures.

SECTION 2
SOFTWARE FACILITIES

GENERAL FEATURES OF SOFTWARE

The system provides a comprehensive array of software to perform multitasking; real-time and data communications applications; and batch, remote batch and data entry processing. The operating system controls execution of tasks and accessing of external devices and files. A complete set of program preparation software is available to develop and debug programs written in COBOL, FORTRAN, RPG or assembly language. An extensive set of utility programs is provided to support program development and execution, and transmission of files from the Level 6 to other computers. System software components are summarized in Figure 2-1.

OPERATING SYSTEM

- MONITOR
- FILE SYSTEM
- PHYSICAL I/O
- COMMUNICATIONS

PROGRAM PREPARATION

- EDITOR
- MACRO PREPROCESSOR
- ASSEMBLER
- FORTRAN COMPILER[a]
- ENTRY-LEVEL COBOL COMPILER[a]
- INTERMEDIATE COBOL COMPILER
- RPG COMPILER
- LINKER
- DEBUG

MOD 400-BES COMPATIBILITY[a]

- EMULATOR
- BUFFER MANAGER

UTILITY PROGRAMS

- COMPARE (LEVEL 6/LEVEL 6)
- COMPARE (LEVEL 6/IBM)
- COPY (LEVEL 6/LEVEL 6)
- COPY (LEVEL 6/IBM OR IBM/IBM)
- CREATE FILE
- CREATE VOLUME (LEVEL 6)
- CREATE VOLUME (IBM)
- DUMP EDIT
- DUMP MEMORY
- DUMP MLCP
- EXPORT PAM FILE
- FILE CHANGE
- FILE DUMP
- IMPORT PAM FILE
- ISL CONFIGURATOR
- LIST CREATION DATE
- LIST IBM CONTENTS
- LIST NAMES
- PATCH
- PRINT/DEFERRED PRINT
- RENAME FILE
- RESET MAP
- RESTORE
- SAVE
- SET DIAL
- SORT/MERGE
- TAPE POSITION
- TRANSMIT FILE
- WALK SUBTREE
- WRITABLE CONTROL STORE

MLCP SOFTWARE

- CHANNEL CONTROL PROGRAMS
- OFFLINE LOADER

SYSTEM CONTROL INTERFACES

- COMMANDS
- OPERATOR COMMANDS
- SYSTEM SERVICE MACRO CALLS

CONFIGURATION

- CONFIGURATION LOAD MANAGER
- HONEYWELL-SUPPLIED SYSTEM

RUN-TIME ROUTINES

- FORTRAN ROUTINES[a]
- ENTRY-LEVEL COBOL ROUTINES[a]
- INTERMEDIATE COBOL ROUTINES

HARDWARE SIMULATORS

- SINGLE PRECISION SCIENTIFIC (SIP) SIMULATOR
- DOUBLE AND SINGLE PRECISION (DSIP) SCIENTIFIC SIMULATOR
- COMMERCIAL SIMULATOR

REMOTE BATCH FACILITY

DATA ENTRY FACILITY

TERMINAL CONCENTRATION FACILITY

IBM 2780/3780 WORKSTATION FACILITY

HASP WORKSTATION EMULATION FACILITY

LEVEL 66 HOST RESIDENT FACILITY

[a]These software components are available only with the SAF version of MOD 400.

Figure 2-1. GCOS Software

The software is available in a SAF (short address form) version, which supports up to 64K words of memory, and a LAF (long address form) version which supports up to 1024K words of memory. Hardware resources associated with this system are described in Appendix C.

INTERFACES TO OPERATING SYSTEM

The software supports the following control interfaces to the operating system:

- o *Commands* submitted by a user to the command processor of the user task group
- o *Operator commands* submitted by the operator to the command processor of the system task group
- o *System service macro calls,* specified in assembly language programs, that invoke Monitor and file system services for user task groups

Command Language

There are five functional categories of commands:

- o To control execution
- o To control directories and files
- o To invoke program preparation software
- o To invoke utility software
- o Interactive commands

Some control functions at the task group level are available through commands. Commands are described in the *Commands* manual.

Commands for Execution Control

Once a task group is created, commands written by the user can be executed under the task group. More comprehensive control of execution is provided to the assembly language program through system service macro calls. Commands are used to:

- o Create then initiate other task groups, or spawn task groups. This provides a multiprogramming capability.
- o Abort or delete a task group, or terminate the task group issuing the request. The abort and delete functions are not available through the batch task group.
- o Create then initiate the execution of a sequence of tasks under a task group, or spawn tasks within a group. Using this capability an application can be executed as a sequence of steps. When the sequencing is done so as to have several tasks active simultaneously, there is multitask execution in one task group.
- o A batch task group cannot create another task group.
- o Control of external switches for intertask communication.
- o List the status of all tasks or open files in a task group.

Commands for Directory and File Control

The file system is based on a tree-structured directory hierarchy. To locate a file, the directory pathname must be known. In order to write programs that are independent of the pathname of the physical file, a program uses a logical file number (LFN). More comprehensive control of directories and files is provided to the assembly language program through system service macro calls.

Commands are used to:

- o Create or release a directory or file
- o List the pathname of the working directory; change the pathname of the working directory
- o Reserve (get) a file for processing (through use of a logical file number).
- o List, in the order searched, the directories that are searched for a given pathname; list file entries in a specified disk directory.

- o Modify the share, read, or write attributes of a disk file.
- o Associate or dissociate a pathname with a logical file number.
- o Remove a file from reserve status.

Commands for Program Preparation

Software to perform program preparation is invoked using a command. Component-specific arguments are provided in these commands. The command name is often identical to the software name, e.g., COBOL, FORTRAN, LINKER, RPG.

Commands for Utility Software Execution

A utility is invoked through a command. The command is often identical to the software name, e.g., PATCH, SORT, MERGE.

Interactive Commands

Interactive commands permit the user to:

- o Establish and terminate access to the system
- o Request execution of a batch task group
- o Send messages to the operator
- o Display the current time

Operator Commands

There are three functional categories of operator commands: execution control; directory, file and device control; and system operation monitoring. Operator commands operate on a task group level and cannot be used to control the execution sequence of tasks in the batch task group or in a user online task group. Operator commands are entered through the operator terminal or read from a command file. A description of the operator commands is found in the *Operator's Guide* manual.

Operator Commands for Execution Control

Initially, operator commands are used to define the operating environment. Subsequently, they can be used to control system operation from the operator terminal. Operator commands are employed to:

- o Create, initiate, abort or delete either a batch or online task group
- o Spawn an online task group
- o Temporarily suspend or reactivate an online task group
- o Temporarily suspend and roll out, or reactivate and roll in the batch task group
- o Load or unload a shareable bound unit from a system memory pool
- o Load assembled firmware files into writable control store (WCS)

Operator Commands for Directory, File and Device Control

Operator commands are used to:

- o Change a system library or working directory pathname
- o Modify the share, read, or write attributes of a disk file
- o Set or delete access (read, write, and execute) to files
- o Set or delete access (list, modify, and create) to directories

Operator Commands to Monitor the System

Operator commands are used to:

- o List all task groups and requests queued for batch execution
- o List the status of all tasks or open files in a task group
- o List the pathname of the working directory
- o List, in the order searched, the directories that are searched for a given bound unit
- o List access specified for files and directories

System Service Macro Calls

System service macro calls are available to the assembly language program to perform a wide variety of Monitor and file system service functions, similar in some instances to those functions accessible through commands. There are two functional categories of system service macro calls: to control execution, and to control directories and files. The macro calls are described in the *System Service Macro Calls* manual.

Macro Calls for Execution Control

Monitor service macro calls are used to:

- Control task groups and tasks
- Manage memory allocation
- Load and execute overlays
- Coordinate the use of resources within an online task group through semaphores
- Control execution based on real-time clock considerations
- Enable or disable user traps
- Display or suppress the display of messages on the operator's terminal
- Designate a task group's command-in, user-in, error-out, and user-out files
- Communicate directly with device drivers to control input/output and devices
- Control external switches for intertask group communication
- Associate or dissociate a pathname with a logical file number.

Macro Calls for Directory and File Control

Monitor service macro calls are used to:

- Create or release a directory or file
- Change or obtain the pathname of the working directory
- Rename a file or directory
- Open, close, get (reserve), or remove the reservation of a file
- Lock/unlock records in a file
- Get information describing a file
- Test the status of an outstanding file activity
- Wait on list until input or output is complete
- Read, write, rewrite, or delete a record of a file
- Read from, or write a block to a file

OPERATING SYSTEM SOFTWARE

The operating system contains software for execution control, the file system, physical I/O, and communications.

Monitor Software

The Monitor contains software to execute requests for Monitor functions and to maintain the control tables that are necessary for the orderly processing of requests. These functions are obtained through commands, system service macro calls, and statements in higher-level languages. Monitor software includes:

- *Task manager* – Handles the disposition of tasks within the system, and responds to requests placed against tasks. It processes requests to activate tasks, returns control to interrupted tasks, and synchronizes, suspends and terminates tasks.
- *Clock manager* – Handles all requests to control tasks based on real-time considerations, and responds to requests for the time of day and date in ASCII format.
- *Memory Manager* – Controls dynamic requests for memory or the return of memory to a memory pool.
- *Trap manager* – Handles the transfer of execution control from an executing program to a predefined trap location when a trap (a special condition such as a hardware error) occurs. The trap manager handles system traps and allows a task group to connect its own trap routines for specific traps.

- o *Operator interface manager* – Manages all messages sent simultaneously by multiple task groups to the operator terminal or from the operator terminal to a task group.
- o *Loader* – Loads the root and overlays of a bound unit dynamically from a disk.
- o *Listener/login* – The listener monitors a selected set of local and remote terminals, reporting any change of state (for example, connect, disconnect) to the login component. If a user submits a login command requesting access to the system, the login component requests that a task be spawned for the user.
- o *Command processor* – Processes all commands. It is the lead task of the batch task group and can be the lead task of an online task group.

File System Software

The file system is based on a tree-structured hierarchy and software functions are provided to create or maintain this directory structure, locate a file by its pathname, create and maintain data files, control concurrent use of files, and provide for the logical transfer of records between an application program and an external device. These functions are available through commands or, for an assembly language program, through system service macro calls, described in the *Commands* manual and *System Service Macro Calls* manual, respectively.

The File System software handles input/output functions of each of the different supported devices, including communications. For disk, it provides four file organizations and access to them; namely, sequential, relative, indexed, and fixed relative file organizations. (Fixed relative file organization is compatible with BES and BES2 files.) Sequential access is provided for magnetic tape, communications, printer, card reader, and terminals. A description of the data file organizations and their properties is found in the *Data File Organizations and Formats* manual.

The languages COBOL, FORTRAN, and RPG use the logical file organizations listed above. The language reference manual for each language provides statements for accessing the logical files.

An assembly language program can access files through file and data management macro calls to the Monitor or through the physical I/O drivers; both methods are described in the *System Service Macro Calls* manual.

The interface to communications software is described in the *Communications Processing* manual.

Physical Input/Output Software

An assembly language program can use physical input/output driver software which works at the hardware physical level. Each peripheral and communications device type has a driver which is a reentrant procedure that can control one or more devices. A description of the peripheral drivers and the physical I/O macro calls is found in the *System Service Macro Calls* manual. Macro calls for communications are described in the *Communications Processing* manual.

Communications Software

Communications software is accessible through the standard input/output interface, is memory and MLCP resident, and interacts with Monitor software to process user communications applications. With the Honeywell-supplied communications software, users need not provide their own communications system programs.

The communications software is user-driven. It answers the phone in response to a user-issued connect; it polls terminals in response to user-issued reads. Users (application or system software) must provide buffers to the communications software to accommodate read and write operations.

Communications software provides a common I/O interface to its users through the standard physical I/O interface (the $RQIO macro call). The communications software components and their functions are summarized below.

- o *Communications supervisor* – Queues user service requests and activates the appropriate line protocol handler, interacts with the user program through system software when

a transaction is complete, and services connect/disconnect requests and line protocol handler timeouts.

- o *Phone monitor* – Provides data set control for detection of phone connect/disconnect, and provides the capability of "hanging up" the phone connection upon user request.
- o *Line protocol handler* (LPH) – Handles error recovery (parity, block control check); initializes the LPH and channel control program; processes interrupts, timeouts, and messages; and handles protocol acknowledgment/negative acknowledgment.
- o *Poller* – Used only for poll and select protocols. Queues poll requests, requests the LPH to poll a terminal, and dequeues the request when the LPH has received data from the terminal.
- o *MLCP driver* – Sets up and processes I/O up to an LPH request, and services MLCP interrupts, passing them to the LPH.
- o *Channel control program* (CCP) – Handles character processing, inserts and deletes protocol headers and framing characters (surrounding a message). An extension of the LPH, the CCP resides in the MLCP and is independent of the central processor; thus character processing overhead is eliminated from central processing.
- o *MLCP macro routines*

For details on communications software functions, line protocol handlers, and the control structures used for communications tasks, see the *Communications Processing* manual.

PROGRAM PREPARATION SOFTWARE

The software in this category allows you to write, compile, link, execute and debug an application program. Each of the program preparation components, except for Debug, is invoked by command described in the *Commands* manual.

- o *Editor* – Used to create and update, on disk, a source unit written in one of the provided programming languages. It will edit characters, expressions, or lines of text. The Editor is reentrant and can support multiple users. A description of the Editor directive language is found in the *Program Preparation* manual.
- o *Macro Preprocessor* – Required to process an assembly language application source unit containing calls to macro routines. A macro routine consists of a specified sequence of assembly language source statements that you want specialized and included in your source module. The Macro Preprocessor creates another source unit with assembly source code replacing the macro calls. A description of the macro preprocessor language statements is found in the *Assembly Language Reference* manual.
- o *Assembler* – Translates assembly source statements of a source unit into text of a relocatable object unit and optionally produces a cross-reference listing indicating symbol usage. The Assembler can process source for the Commercial Control Processor Models and the Scientific Instruction Processor (SIP). The Assembler supports coding of user-defined generic instructions to be executed on the Writable Control Store (WCS). Writable Control Store (WCS).
- o *FORTRAN Compiler* – Translates FORTRAN source statements of a source unit into text of a relocatable object unit and source listing or optionally, assembly language source statements in a source unit. The language is based on the American National Standards FORTRAN 77 subset. Offered in the language are the Instrument Society of America (ISA) extensions for bit string manipulation and task management, and a Honeywell extension for communications. FORTRAN programs and Entry-Level COBOL programs can call each other. FORTRAN is intended for commercial and scientific application programming. A description of the FORTRAN language statements is found in the *FORTRAN Reference* manual.
- o *Entry-Level COBOL Compiler* – Translates source statements of a source unit into text of a relocatable object unit. Entry-Level COBOL programs and FORTRAN programs can call each other. Significant features of Entry-Level COBOL include: file handling for sequential, relative, and indexed files; three-dimensional tables and indexing; CALL/CANCEL capability; DISPLAY and COMP-1 data; full American National Standards

editing; 21 verbs; and communications through file management facilities. For descriptions of the Entry-Level COBOL language statements, refer to the *Entry-Level COBOL Reference* manual.

- *Intermediate COBOL Compiler* – Translates source statements of a source unit into text of a relocatable object unit. Intermediate COBOL supports Entry-Level COBOL features plus additional features. Descriptions of the Intermediate COBOL language statements are given in the *Intermediate COBOL Reference* manual.
- *RPG Compiler* – Translates RPG source statements of a source unit into a set of object units consisting of a root, or a root plus multiple overlays. The compiler also produces a file containing Linker directives; user-written Linker directives are thus unnecessary. When the command processor is invoked to process the statements in this file, it invokes the Linker, and supplies it with Linker directives necessary to create an executable bound unit. The compiler supports an RPG language comparable to that in current industry-wide use. Significant features include: look-ahead, control levels and matching fields on input; table and array processing; forms alignment; and editing, detail, and total time functions on output. The compiler generates Commercial Instruction code. A description of the RPG language is found in the *RPG Reference* manual. *
- *Linker* – Combines object units that are the output of a compiler or the Assembler and produces a bound unit for subsequent loading. It resolves external references made between object units being linked. Linker directives can be used to create reentrant bound unit files. A description of Linker directives is found in the *Program Execution and Checkout* manual.
- *Debug* – Used for testing programs at the machine language level. Hexadecimal patches can be made to the program. Debug is invoked as a separate task group within the system. A description of the Debug directives is found in the *Program Execution and Checkout* manual.

*

UTILITY SOFTWARE

A comprehensive set of utility programs is available to support file management and program development. All utility programs listed below are invoked by commands except for Memory Dump (MDUMP) and Dump MLCP (DUMCP). The usage of the utility programs is described in the *Commands* manual unless otherwise indicated.

- *Compare* – Compares two volumes, files or portions of files for equality, and lists the discrepancies.
- *Copy* – Copies a file or volume. Logically reorganizes the file. Copies can be placed on tape or disk.
- *Copy IBM* – Copies Level 6 diskette files to IBM diskette files (and vice versa); also copies one IBM volume to another IBM volume.
- *Create IBM Volume* – Formats an IBM 3740 diskette.
- *Create Volume* – Creates or modifies a volume. Formats and labels a disk or tape volume, creates disk bootstrap records, or renames a disk volume.
- *Dprint* – Prints the contents of the specified file. By use of Dprint, the print request is entered in a queue, and the user can continue with other commands or terminate the session (logout).
- *Dump Edit* (DPEDIT) – Produces an edited logical or physical dump image of memory, or edits and prints out a disk file containing a dump of main memory that was obtained through the MDUMP bootstrap record. (Described in the *Program Execution and Checkout* manual.)
- *Dump Memory* (MDUMP) – Dumps the contents of memory to a disk file when a program aborts or halts, by using the bootstrap record MDUMP on a specially created disk volume. The Dump Edit utility is then used to print the dump. (Described in the *Program Execution and Checkout* manual.)
- *Dump MLCP* – Dumps contents of all or part of Multiline Communications Processor (MLCP) memory. (Described in the *Communications Processing* manual.)

- o *File Change* – Changes the contents of a disk sector or control interval.
- o *File Dump* – Performs both logical and physical dumps from disk or 9-track magnetic tape; performs physical dump only from 7-track tape; output in both alphabetic and hexadecimal notation.
- o *Import/Export PAM File* – Converts members of GCOS/BES partitioned files to and from GCOS 6 variable sequential files; used to transport programs between BES and GCOS 6.
- o *ISL Configurator* – Reads ISL (Intersystem Link) directives from a user input file and generates an ISL loader.
- o *List Creation Date* – Lists creation dates of files in a directory.
- o *List IBM Contents* – Lists the contents of an IBM diskette.
- o *List Names* – Lists the file and/or directory entries contained within the specified directory.
- o *Patch* – Applies hexadecimal patches to an object unit or bound unit. Provides a facility for program correction without recompilation or reassembling. (Described in the *Program Execution and Checkout* manual.)
- o *Print* – Prints the contents of the specified file. Unless specified otherwise, the file is written to the current user_out file. The user must wait for the print to complete before entering another command.
- o *Rename File* – Assigns a new name to an existing file or directory.
- o *Reset Map* – Lists the number of logical sectors available for allocation on a disk volume.
- o *Restore* – Restores the file structure and physically reorganizes the disk volume previously saved (by the SAve command).
- o *Save* – Saves the file structure and data of a disk volume onto a tape or disk file.
- o *Set Dial* – Sets a phone number for subsequent use by a named channel/communications terminal file.
- o *Transmit File* – Supports file transmission between the Level 6 system and other Level 6 processors, or between the Level 6 and any of the following host processors: Level 62, 64, or 66; Series 200/2000; or non-Honeywell systems that use the BSC 2780/3780 protocol. Three utility programs, described in Section 7, provide the file transmission capability.
- o *Walk Subtree* – Executes specified command in specified directory and in all subordinate directories.

Sort/Merge

Sort and Merge are invoked by separate commands. Sort may also be called from a COBOL, FORTRAN, or assembly language program. The Sort program arranges records of a file in an order based on the values of user-specified record key fields. Merge combines the records of up to six sequentially ordered input files on the basis of record key values. Up to 16 key fields can be specified, with values to be arranged in ascending or descending order according to the ASCII collating sequence. The data type of a key field can be character string, signed binary, packed decimal, or signed/unsigned unpacked decimal. Sort/Merge options include record selection, redefinition or rearrangement of record contents, and deletion of duplicate records. See the *Sort/Merge* manual for a detailed description of these capabilities.

RUN-TIME ROUTINES

Run-Time I/O Routines

The FORTRAN run-time I/O routines provide for data transfer, peripheral or communications device manipulation, and the processing of data as specified in FORTRAN FORMAT statements. These routines use the file system to accomplish open, close, and position file functions, and to read and write formatted and unformatted records. They contain data conversion routines to edit integer, real, logical, and character data for formatted input and output. Only those routines required by a particular FORTRAN program are linked to form the bound unit.

The COBOL run-time I/O routines provide a logical I/O interface for the transfer and processing of data at program execution time. The routine is linked with the program's object unit, and uses the file system to open, close, and position files and to read and write records to peripheral or communications devices. Separate run-time routines are provided for Entry-Level and Intermediate COBOL.

The FORTRAN and COBOL routines produce diagnostic messages to inform the programmer of inappropriate or inconsistent input/output statements.

FORTRAN Run-Time Routines

The software includes FORTRAN mathematical and bit string manipulation routines. These intrinsic functions are available in object module format, linked on an as-needed basis to perform a variety of operations on behalf of a FORTRAN program. Optionally, they can be loaded during configuration as an operating system extension available to all online applications. Some of the operations performed by these routines are:

- o Date and time subroutines
- o Converting to and from integer and real values
- o Truncation
- o Determining the nearest whole number
- o Transferring a sign
- o Choosing: the largest value; the smallest value
- o Finding: the length of a character entity; the square root; the natural logarithm; the common logarithm
- o Computing selected plane and spherical trigonometric functions
- o Performing bit string manipulation operations on integer data: inclusive OR, exclusive OR, products, complement, shift, clear or set a bit, and test a bit value.

FORTRAN routines are available to implement the management of tasks. Functions are provided to:

- o Initiate a task after a designated period of time
- o Suspend a task

Communications programs are provided with two routines, ZFSTIN and ZFSTOT, to test the status of the system buffer prior to issuing a READ or WRITE. Depending on the return status, the FORTRAN program can loop on the test, place itself in the wait state, continue other processing, or issue a READ or WRITE and stall if the I/O buffer is busy.

See the *FORTRAN Reference* manual for details about these routines.

HARDWARE SIMULATORS

The SSIP and DSIP (single- and double-precision scientific instruction processor) provide software simulation of

- o Floating-point instructions (add, subtract, multiply, divide, compare, load, store, swap, and negate) that are generated by the FORTRAN Compiler or the Assembler.
- o Floating-point branch instructions (branch on bit settings of scientific indicator register or scientific accumulator values).

The Commercial Simulator provides software simulation of commercial instructions (commercially oriented calculations and operations) that are generated by the Intermediate COBOL Compiler, RPG Compiler, or Assembler.

CONFIGURATION LOAD MANAGER

The Configuration Load Manager (CLM) accepts configuration directives from either a Honeywell-supplied input file or a user-generated input file (CLM_USER) to perform system configuration. Configuration directives are available to:

- o Define system variables (e.g., real-time clock, scientific and commercial processors)
- o Describe peripheral devices and their characteristics
- o Define system, batch, and one or more online memory pools
- o Identify system software and application-specific bound units that are to be permanently resident in the system area of memory
- o Define the communications environment of the operating system

Configuration procedures are summarized in Section 4. A complete description of configuration directives appears in the *System Building* manual.

BES-MOD 400 COMPATIBILITY

The GCOS 6 MOD 400 Monitor and I/O system services are a superset of the GCOS/BES online Executive system services. However, differences exist in:

- o Assembly language programs containing calls to the BES Executive
- o BES object modules that must be imported to execute on MOD 400
- o BES COBOL programs that require the BES COBOL and run-time routines for MOD 400 execution
- o Configuration commands

Appendix A of this manual describes the procedures to be used to convert and execute BES programs under MOD 400.

Disk Device Files – The general form of a disk device-level access pathname is:

>SPD>dev_name[>volid]

where dev_name is the symbolic name defined for the disk device during system building, and volid is the name of the disk volume.

This pathname format is used only when access to the entire volume is required, e.g., during a volume copy or volume dump.

If the volid is not supplied, reservation of the disk is exclusive; i.e., the reserving task group has read and write access, but other users are not allowed to share the file. This pathname form is used when creating a new volume.

Relative Pathname	*Full Pathname*
DELTA	^ SYS01>UDD>PROJ1>USERA>DELTA
OLD>DELTA	^ SYS01>UDD>PROJ1>USERA>OLD>DELTA
<USERB>ALPHA	^ SYS01>UDD>PROJ1>USERB>ALPHA
<<PROJ2>USERA>DELTA	^ SYS01>UDD>PROJ2>USERA>DELTA
<	^ SYS01>UDD>PROJ1

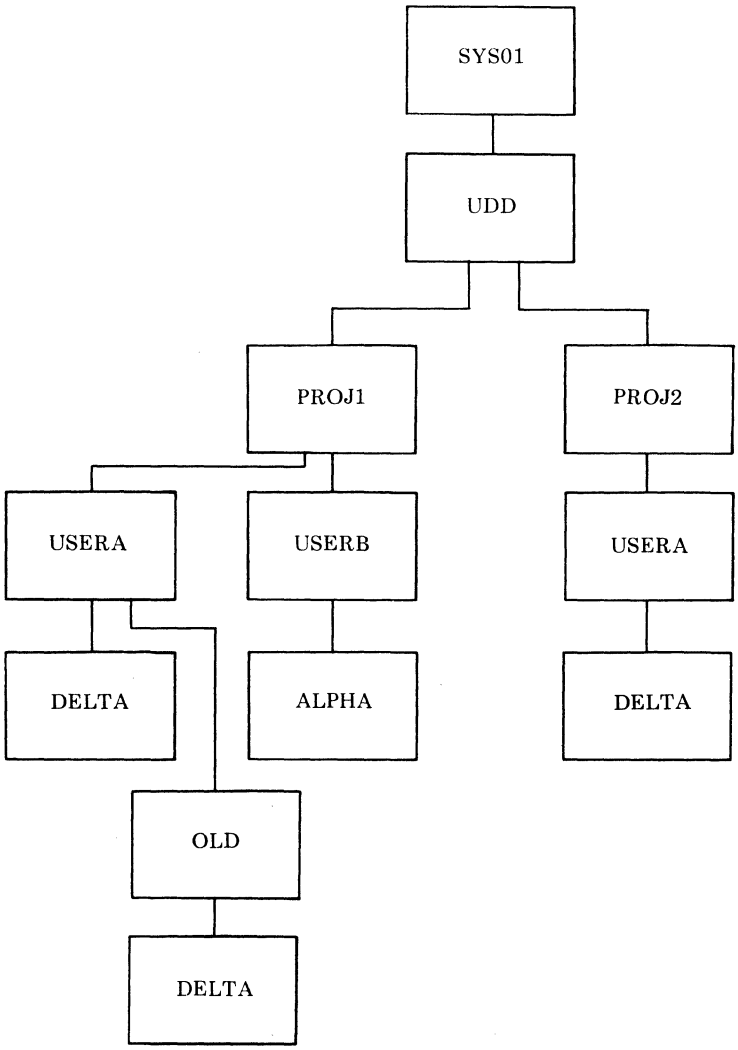

Figure 3-1. Sample Pathnames

If the volid is specified, reservation is read/share; i.e., the reserving task group has read access only, other users may read and write. This pathname form is used when dumping selected portions of a volume without regard to the hierarchial file system tree structure.

The following are examples of device pathnames:

Peripheral Device	*Pathname*
Line printer	>SPD>LPT01
Exclusive tape volume	>SPD>MT902>VOL3
File on exclusive tape volume	>SPD>MT902>VOL3>FILEA
Exclusive diskette	>SPD>DSK02
Nonexclusive disk volume	>SPD>RCD01>V23X

Special Pathname Conventions

Star names and equal names are special pathname conventions available for use with certain utility programs. They provide shorthand methods of specifying a related set of entry names to commands.

Star Convention

A star name is an entry name that identifies a group of entries within a single directory. It is composed of one or more nonnull components, one of which consists of an asterisk (star), and can contain up to 12 ASCII characters, none of which can be the greater than (>), less than (<), or circumflex (^) character.

A star name can be used only as the final entry name of an input pathname for the following commands:

- o COPY
- o COMPARE
- o LIST NAMES
- o LIST CREATION DATE
- o RENAME

These commands perform their function for each entry identified by the star name.

A star name identifies all directory entries having an entry name that matches the star name. Each component, except the last, of a star name ends with a decimal point. A component position corresponds to the number of components from the left; i.e., the leftmost component is in the first component position. A special type of matching is performed in which components of a star name that do not contain an asterisk are compared with corresponding components of an entry name, while other entry name components are ignored. Entries identified by a star name all have similar names in that they are all determined by the star name template. For example, the star name *.IN.A refers to all three-component entry names ending in .IN.A (Macro Preprocessor include files) in the working directory.

An asterisk may appear in any component position of an entry name; each asterisk is treated as a special character. Each asterisk character used in a star name designates any number of characters (including none) appearing in the corresponding component position of the entry name. A double asterisk (**) may be used to indicate any number of contiguous component positions within an entry name. One or more single asterisks may appear in a star name; only one double asterisk component may be used. For example, the star name *.MY_PROG.** identifies all multiple-component file names, the second component of which is MY_PROG, in the working directory.

A question mark character (?) may be used within a star name to match any character position that appears in the corresponding component and character position within that component of the entry name. Multiple question marks may be specified, each representing exactly one character position. For example, the star name *.MY_?????.* identifies all three-component file names, the second component of which is an eight-character name beginning with MY_, in the working directory.

For complete details concerning the star convention, refer to the *Commands* manual.

Equal Convention

Three commands (i.e., COPY, RENAME and COMPARE) that accept pairs of pathnames as their arguments allow the final entry name of the first pathname (input) to be a star name and the final name of the second pathname (output) to be an equal name. An equal sign as a component of the output pathname means that the character string from the corresponding component of the input pathname is to be substituted for the equal sign. Use of this convention allows the user to copy or compare multiple input files without specifying complete individual names. Output file pathnames are determined by the equal name convention after the input file pathnames containing star names have been resolved.

The equal names convention provides a powerful mechanism for mapping certain character strings from the first pathname into the second pathname of the pair. Such a mechanism helps to reduce typing required to specify the second pathname, and it can be essential for mapping character strings from the entry names identified by the star name into the equal name, because these character strings are not known at the time the command is issued.

Under the equal convention, the mapping of character strings from the star name into the equal name is performed according to rules for constructing and interpreting equal names. An equal name is an entry name composed of one or more nonnull components, one of which consists of an equal sign, and can contain up to 12 ASCII characters, none of which can be the greater than (>) or less than (<) characters. An equal sign can appear in any component position of an entry name; each is treated as a special character. The equal sign represents the corresponding component of an entry name identified by the star name. An error occurs if the corresponding component does not exist. A double equal sign (==) component can be used to represent all components of entry names identified by the star name that have no other corresponding components in the equal name.

The percent sign (%) functions in the same manner as the question mark in a star name. The percent sign can be used within an equal name to match any character that appears in the corresponding component and character position within that component of the entry name identified by the star name. An error occurs if the corresponding character does not exist or if an equal sign appears in a component that also contains a percent sign. Multiple percent signs can be specified, each one representing exactly one character positon.

For example, the command

```
COPY R DATA_BASE S =
```

creates a duplicate copy of the input file in the working directory, but assigns the name S.DATA_BASE to the duplicate file.

Likewise, the star convention can be used to address all files with a specific component name in the working directory. The command

```
COPY * DATA_BASE = DATA
```

copies all two-component entry names with DATA_BASE as the second component and assigns the name DATA as the second component of the newly created files.

DATA FILE ORGANIZATIONS AND ACCESS

Disk File Organizations

UFAS Sequential – Logical records are normally read from or written to a sequential file in consecutive order. Records must be written sequentially although the file can be positioned for writing by specifying a simple key. Records can be read, modified, or deleted directly by specifying their exact control interval and record address (simple key). Records cannot be inserted but they can be appended to the end of a file. Fixed or variable length records may be used. The record size can exceed the control interval size. If a record is deleted, the position it occupied cannot be reused.

UFAS Relative – A UFAS relative disk file can contain fixed or variable length records. If variable length records are used they occupy fixed length slots, and the size of the largest

record must be specified. A record can be updated (i.e., rewritten), deleted, or appended to the file. If a record is deleted, the position it occupied can be used for a new record. A file can be created directly by specifying relative record numbers in random sequence. When writing records, the user has the option of letting data management search for the next available location.

UFAS Indexed – Each logical record contains a key field of fixed size that occupies a fixed position. Records are logically ordered by key value and can be accessed sequentially in key sequence or directly by key value. Fixed- or variable-length records may be used. Variable length records are handled in variable-length format. A record can be updated, deleted, or inserted in key sequence into available free space. When no space is available to insert a record in key sequence, the record is placed in an overflow area. When the file is initially loaded, the records must be supplied in ascending sequence by key value.

Fixed Relative (BES Compatible) – Only fixed-length records are supported. Records are accessed sequentially or directly by their position relative to the beginning of the file. Records can be updated or appended to the file. A file can be created directly by supplying relative record numbers in random sequence. When writing records, the user has the option of letting data management search for an available location. Two types of files are supported: those that allow deletable records and those that do not allow deletable records. If deletable records are allowed, a deleted record position may be used for new record data. If deletable records are not allowed, new records can be inserted anywhere in the file. Fixed relative files are designed to be used with BES1 or BES2 Executive system. A fixed-relative file is incompatible with file organizations on other GCOS systems.

Tape File Organization

Fixed- or variable-length records may be used. Records cannot be inserted, deleted, or modified, but they can be appended to the end of the file. The tape can be positioned forward or backward any number of records.

A block can be treated as one logical record called an "undefined" record. An undefined record is read or written without being blocked or unblocked or otherwise altered by data management.

A labelled tape is one that conforms to the current tape standard for volume and file labels issued by the American National Standard Institute. The following types of labelled tapes are supported:

single-volume, single-file
multi-volume, single-file
single-volume, multi-file
multi-volume, multi-file

The following types of unlabelled tapes are supported:

single-volume, single-file
single-volume, multi-file

Data File Access

The languages COBOL, FORTRAN, and RPG use logical file organizations as described above. Refer to the language reference manuals for the relationship between a language's logical file and the system's physical file organization. Each language provides statements for manipulating files. Files can be manipulated by assembly language programs through File and Data Management macro calls to the Monitor or through the physical I/O drivers; both methods are described in the *System Service Macro Calls* manual.

Any element of the file system can be protected from unauthorized access or use. Access to a file or directory can be limited through establishment of an *access control list* for the file or directory. Data integrity of shareable files can be protected through specification of concurrent access privileges to particular task groups requesting use of the file. The two control techniques are described in this section.

Access Control Lists

The system offers an optional means of preventing accidental or unauthorized access to data through the directory structure. The access rights to files and directories can be controlled through access control lists established by individuals responsible for the files or directories.

Note:
The setting of access rights to system files/directories or other files/directories intended for common use is the prerogative of system administrators.

An *access control list* (ACL) for a file or directory contains entries for all acceptable users, each entry identifying a user (or class of users) and the access rights granted. The user identifier stored in the ACL entry consists of the same components (person.account.mode) as the identifier under which the user logs in. Conventions are available to designate classes of users (for example, all users under a specific account) in ACL entries.

A user is allowed to access a file or directory in a particular way only if the identifier under which he logged in and the type of access he requests correspond to appropriate entries in the applicable ACL.

To grant access rights to all files or directories described in a particular directory, a common access control list (CACL) can be established for the governing directory.

Each time a user requests access to any of the files or directories described in that directory, the applicable ACL and CACL are both checked. Conflicts between ACL entries and CACL entries that both pertain to a specific user are resolved through a priority matching scheme. Use of CACLs permits users or classes of users to be authorized access to collections of files or directories without requiring that each individual ACL be changed.

Access rights that can be granted in ACLs and CACLs include the following: road access, write access, and/or execute access for files; list access, modify access and/or create access for directories. Users can also be denied access to a directory or file.

File Concurrency

Concurrent read or write use of a file is established by the task group that reserves the file. Concurrency has two aspects: (1) it establishes how tasks in the reserving task group intend to access the file, and (2) it establishes what the reserving task group allows other task groups to do with a file. If the file is already reserved, a task group's concurrency request will be denied when its intended access conflicts with the access permitted by another task group. The concurrency request will also be denied if what it allows others to do conflicts with the access already established by another task group. For example, if a task group reserves the file exclusively, other task groups are denied access. Of if a task group permits read only access but does not permit write access, other readers are allowed but writers are denied access.

Concurrency is controlled through the GET command or through the $GTFIL system service macro call. The possible combinations of access intended for the reserving task group and the sharability permitted other task groups are given in Table 3-1.

TABLE 3-1. DISK FILE CONCURRENCY CONTROL

Reserving Task Group	Other Task Groups
Read only	Read only (Read share)
	Read or Write (Read/write share)
Read or Write	No read, no write (Exclusive use)
	Read only (Read share)
	Read or Write (Read/write share)

System File Concurrency

Compiler generated programs, commands, Sort, and other system software always request exclusive concurrency for files that they reserve for a user. The operator terminal must be reserved with read/write shared concurrency to allow concurrent access by

many task groups. For this reason, the command argument -COUT specifying the list output file cannot be the operator terminal. If the command-in and user-in files are on disk, they are reserved with read-only shared concurrency; if assigned to a user terminal, they are reserved with exclusive concurrency. The user-out and command-out files are always reserved for exclusive use.

Record Locking (Shared File Protection)

The record-locking facility is an optional file access feature that provides protection of and controls contention for records within shared disk files. It is designed primarily to prevent interference in applications where many users perform short transactions on common files. If locking is to be implemented, a lock pool must be created at system configuration. (See CLM command RLOCK in the *System Building* manual.)

Locking is performed on a file by a specific request when the file is reserved. This is accomplished by a $GTFIL system macro call or a GET command that includes a _LOCK argument.

Record locking provides a shared-read, exclusive-write ability. The first user to access (read or write) a record locks the record and prevents other users from writing into it. An attempt to write into a locked record causes the system to note the interference and to issue an error return code. When a record is locked, the entire control interval (CI) in which it is contained becomes locked.

For record locking purposes, a logical file number (LFN) within a task group uniquely identifies a user. Thus this facility prevents a record from being concurrently written by different task groups. Moreover, a task group can be created so that different tasks within it use different LFNx to access a given file. In this way, record locking can be used to eliminate interference between tasks of a task group.

FILE SYSTEM BUFFERED OPERATIONS

A buffer is a storage area used to compensate for a difference in rate of flow of data, or time of occurrence of events, during transmission of data from one device to another. As used in I/O programming, the term buffer refers to an I/O area in systems that provide the possibility of I/O overlap. Buffering is the process of allocating and scheduling the use of buffers. In sequential data processing, for example, overlap of input operations and processing can be achieved by anticipatory buffering; i.e., the next block is read into memory before it is needed. The program can then process records from block n while block n+1 is being read into memory.

This system supports two types of buffered operations: one for unit record and terminal devices, the other for disk and magnetic tape devices.

Unit Record and Terminal Buffered Operations

All printers and *most* interactive terminals are provided with one File System buffer. To provide a system buffer for the card reader or terminals configured as file types KDL, TDH, or TDL, the B parameter in the CLM DEVICE directive must be specified. The operator terminal (system LRN 0) cannot be buffered. By providing a File System buffer, asynchronous I/O can be done; i.e., application code can execute in parallel with I/O transfers.

All terminals (except the operator's) and printers, except file types KDL, TDH, and TDL, have tabbing capability through software that converts the tab into spaces. Default tabulation stops are set at position 11 and every tenth position thereafter for the line length of the device.

Asynchronous I/O operates in two different ways, depending on whether data is obtained from a device (reading) or transferred to a device (writing).

Buffered Read Operations

An application task issues a logical READ to a File System buffered device. If the buffer is full from a prior anticipatory read, the data in the buffer is transferred into the application task's area; then a physical I/O transfer into the system buffer (an anticipatory read) is performed in parallel with continued task execution. If the buffer is not full, task execution stalls until the anticipatory read is completed.

The timing of the initial anticipatory read performed for the card reader is different from that of the interactive terminals; afterwards it is the same. An application task issues an OPEN call to the card reader. Immediately after the OPEN is complete the FILE System performs an asynchronous anticipatory read into the system buffer while the application continues execution. All OPEN calls are synchronous.

For interactive terminals, immediately after the OPEN is complete an asynchronous physical connect is performed while the application continues execution. Assembly or FORTRAN applications can check the status of the OPEN to see if a READ can be issued without stalling application execution. File System issues an asynchronous anticipatory physical read when a status check after the physical connect is complete. The file status remains busy until the physical read is done and the system buffer is full. At this point, the file status is "not busy" (i.e., the anticipatory read is successfully completed), and the application can issue a READ with the assurance of receiving data immediately. If at any point after the OPEN is issued the assembly or FORTRAN application issues a READ before the physical connect and anticipatory read have been completed, the READ is synchronous and further central processor execution is stalled on this application until the anticipatory read is complete. To avoid status check looping to test the input buffer status or stalling on a READ, both assembly and FORTRAN applications can put themselves into the wait state, thus making the central processor available for lower priority tasks. After the OPEN, a COBOL application must issue READ requests. The COBOL application will be put in wait state if it is executing its I/O statements in synchronous mode. Otherwise, the COBOL run-time package performs the status checks and returns a 9I status until successful completion. The COBOL program can either loop on the READ or continue other processing.

The anticipatory read allows an application to control input from more than one interactive terminal, each of which represents a data entry terminal. By testing the status of the system buffer before a READ (FORTRAN, assembly) or by checking for the 9I status return after a READ (COBOL) even if a terminal operator is not present at the time of the READ request, the application will not be stalled and it can continue to poll other terminals.

Buffered Write Operations

A buffered write operation to a device works on behalf of the application program in the same logical manner as the read – the program is permitted to execute in parallel with the physical I/O transfer to the device. To achieve this parallel processing, no special operation occurs on an OPEN call and no distinction is made between interactive and noninteractive file types. Each WRITE call is completed by moving data from the application buffer to the File System's buffer (performing any detabbing, if requested), initiating the transfer, and returning control to the application program. If the program performs a second WRITE while the system buffer is still in use for a previous transfer, the application is stalled until the buffer is available and new data moved into it again. The application can avoid stalling execution by checking the status of the system buffer before issuing a WRITE to an interactive terminal to see if, in a special mode, it is still in use or not (FORTRAN, assembly) or by testing for the 9I status return after the WRITE (COBOL, for interactive devices only).

If a WRITE call is issued while data is being entered (because of a read) into the system buffer, the read is allowed to complete and input data is saved in the system buffer, a synchronous write is reissued by File System, and output data is transferred directly from the application buffer. However, tab characters are not expanded into spaces by software.

Special considerations for buffered write operations arise because, if a physical I/O error occurs while data is being transferred from the system buffer to the device, the application program must be aware that the error occurred on the previous write operation. Furthermore,

if an error does occur, the application program may need to have saved (or be able to retrieve) the data record so that it can be repeated.

Disk and Magnetic Tape Buffered Operations

An assembly language application can request buffers through the system service macro call to get a file for reservation. If File System needs buffers for blocking or unblocking, it provides either the number of buffers specified or, by default, one buffer. If no buffers are needed, none are provided, even if specified. Each buffer contains a disk control interval (CI) or magnetic tape block. When an application program issues a READ and the desired record is not in any buffer, the next empty available buffer is filled with the CI or block containing the record; when all buffers are filled, an active buffer is selected for the next different CI or block based on a least recent usage algorithm.

I/O assembly language macro calls for disk or magnetic tape operations are synchronous, and the application stalls until the I/O operation is completed. Asynchronous I/O can be obtained through File System Storage Management macro calls and, optionally, through physical I/O (PIO).

DEFERRED PRINTING

The deferred printing utility allows output streams to be printed concurrently with job execution. The following procedure can be used to implement deferred printing.

1. Define an online task group whose function is to produce print output files; for example

   CG $P 0 _EFN XPR _POOL AB

   XPR specifies the deferred print utility and uses printer>SPD>PRINTER or >SPD>LPT00, whichever is configured.
2. When a print file is to be printed, issue the following command:

   DP file-name [ctl arg]

   file-name – May be specified by a simple or absolute pathname
   [ctl arg] – Optional arguments that allow various characteristics (e.g., line length, headings) to be specified. These arguments are described in the *COMMANDS* manual.

When step 2 is completed, a request is issued to system task group $P, and the user's group may then accept other tasks or be logged out.

SECTION 4

SYSTEM ACCESS

SYSTEM CONFIGURATION AND ENVIRONMENT DEFINITION

At larger installations a system programmer might design the configuration files and the possibly different operating environments to be used at the installation. The daily startup would be done by an operator. At smaller installations, especially those where programmers run dedicated applications, each programmer might do the configuration and startup for his application.

Creation of a usable system consists of a two-step procedure:

- o Bootstrap a Honeywell-supplied system startup routine that provides a limited operating environment for building the files used in the second step
- o Specialize the system startup procedure by configuring a system to correspond to the installed hardware and by defining the environment in which to prepare and execute applications programs

The bootstrap operation simply consists of turning on the power supply to the hardware, mounting the cartridge disk or diskettes containing the MOD 400 operating system software, and pressing several control panel keys including bootstrap load to execute a standard bootstrap routine. The bootstrap operation includes the initial configuration and startup operations; procedures are executed (1) to configure a limited system consisting of cartridge disks storage modules, or diskettes, and operator terminal, and (2) to provide a one-user online environment that can be used to specialize system startup, perform program preparation, or perform application program execution.

In the provided user environment, the user can employ the Editor to create two files that specialize system startup:

- o CLM_USER – Contains configuration directives, which when executed, will configure a system to correspond to the actual installation hardware
- o START_UP.EC – Contains operator commands, which when executed, will define the installation-specific operating environment consisting of task groups

When these files have been created, the system is again bootstrapped. However, this time directives in the CLM_USER file control the configuration, and operator commands in the START_UP.EC file define the operating environment. (Refer to the *System Building* manual for complete details.)

Configuration directives are available to perform the following functions:

- o Describe available central processing unit options, such as the real-time clock, scientific processor, additional overlay areas, and trap save areas.
- o Describe peripheral and communications devices and their characteristics.
- o Specify the memory pools that partition memory. The system and each user task group, under which applications execute, must be associated with a single memory pool. System and user memory pools are discrete.
- o Indicate which operating system overlay areas should be permanently resident.
- o Indicate that an application-specific bound unit should be permanently resident and be part of the operating system.

The START_UP.EC file is described later in this section.

ACCESSING THE SYSTEM

Ways to Access the System

An installation can simultaneously support several ways to access a system, so a user must determine the access available at a terminal. For simplicity of discussion, three types of access will be described. Access can be (1) through a LOGIN command, (2) through operator control, or (3) through a user's own applications design.

Logging In

The login function allows a user, without operator intervention, to activate an application from any one of the designated terminals. The login function can be activated by the operator after configuration is complete as described in the *Operator's Guide.* Depending on the capability designed and configured for an installation, a user can login in one of three ways. A user can:

1. Type in a LOGIN command as described in the *Commands* manual.
2. Type in the abbreviation of a specific LOGIN command line. A file contains, for each abbreviation, an image of the LOGIN command line that can either be used at all login terminals or that might be restricted to designated ones.
3. Turn on the terminal and be logged in through a direct login. Direct login is useful for transaction processing applications where the user wants to interact with the application and not the operating system, e.g., an application to provide, on request from the designated direct login terminals, the current inventory of a product.

The Secondary User

The system contains a capability that allows a user to login as a secondary user of an existing task group. In order to achieve this capability, the following functions must be performed:

1. A running task group must request the use of a terminal as a secondary user terminal. The request for a secondary user terminal is entered by means of a Request Terminal macro call. (The Request Terminal macro call is described in the *System Service Macro Calls* manual.)
2. The user must login as a secondary user. The login command contains an optional argument that permits a user to login as a secondary user. (The format of the login command is described in the *Commands* manual.)

In an applications program, the user is permitted to specify multiple Request Terminal macro calls. Each Request Terminal macro call will accept a particular terminal that can be used as a secondary user terminal. The number of Request Terminal macro calls to be issued by a task group and the amount of elapsed time likely to occur before each request is "answered" by a secondary login are factors the applications programmer must consider when reviewing design objectives. Whatever cooperation is to exist between task group and secondary users is entirely the responsibility of the applications program designer.

Operator Assigned Access

The operator or another user must activate the application that is to be run and also designate the terminal that is to be used to input commands or user input required by the program executing the command. Terminals that are used for logging in cannot be assigned by the operator or another user. An installation can have a mixture of terminals: some that may be used for logging in and others that may be assigned through the operator or another user.

User Designed Access

A user at an installation that allows use of the system for a single dedicated application, must configure and startup the system, act as operator, determine what the application

environment should be, and how to access the system for that application. If an installation has one terminal, it is used both as an operator terminal and user terminal, as described in the *Operator's Guide.*

The Activated Lead Task

When a user successfully gains access to the system, executable code for the lead task (i.e., the controlling task of the application) is loaded and activated. The lead task can be designated to be either the command processor or a user application. When the command processor is the lead task, the user has complete flexibility to control execution by being able to execute any command in the *Commands* manual. When an application is the lead task, the command processor is not part of the task group.

COMMAND ENVIRONMENT

The command environment is that environment in which the user can communicate with the operating system through the use of command lines entered at a terminal or read from a command file. The essential parts of the command environment, from the user's point of view, are the command processor and the command input file (command-in). The command processor is the system software component which reads command lines issued by the user. It interprets them into procedures that load and initiate execution of bound units which fulfill the requests represented by the command lines. The command input file (command-in) is the file from which the command lines are read. It can be a terminal device, as in the case of an interactive user, or a command file stored on disk or on cards, as in the case of a noninteractive user.

Three other files are involved with, but not limited to, the command environment. These are the user input file (user-in), the user output file (user-out), and the error output file (error-out).

The user-in file is the file from which a command function, during its execution, reads its own input. When a task group request has been processed, and as long as no alternate user-in file is specified as an argument in a subsequent command, the user-in file remains the same as the command-in file. At the termination of a command which names an alternate user-in file, the user-in file reverts to its initial assignment. The directives submitted to the Editor following the entry of the EDITOR command, for example, are submitted through user-in. No specific action is required on the user's part to activate, or to connect to, user-in unless the directives are to be read from a previously created disk file. The user simply invokes the Editor and begins entering editor directives through the same terminal; the attaching of the terminal to the user-in file is invisible to the user.

The user-out file is the file to which a task group normally writes its output. However, certain system components (compilers, etc.) also write to list files (path.L) or to the output file defined in the -COUT argument. The user-out file is initially established by the -OUT argument of the EBR, EGR, or SG command. (Thus, originally, it is the same device as the error output file device.) It can be reassigned to another device by use of the FO (file out) command or by the use of the $NUOUT (new user out) system service macro call. Such a reassignment remains in effect for the task group until another reassignment occurs. Again using the Editor as an example, any responses from the Editor, such as the printing of a line of the file being edited, are issued through user-out. As in the case of user-in, no special action is required of the user to attach his terminal to the user output file. The only time such action would be required is if the output from the command were to be directed to some device other than the terminal.

Error-out is used by the system to communicate to the user an error condition which may be detected during the interpretation of a command or its subsequent execution. Such a condition could be a missing command argument, reported by the command processor, or a file not found condition, reported by the invoked command. The error output file is the same as the initial user-out file. The user cannot reassign error-out through a command, only through a $EROUT system service macro call.

Subsequent paragraphs in this section describe in detail the functions available to the user at command level.

COMMAND LEVEL

When the system is in a state capable of accepting a command from command-in, it is said to be at command level. The methods whereby command level is achieved and the functions that the system performs while at command level are described in the following paragraphs.

Achieving Command Level

Command level can be achieved in any of several ways. Regardless of the way in which the system arrives at this state, the system indicates that it is at command level by issuing a "ready" prompter message at the user's terminal. (This assumes that the user has not disabled the ready message by issuing a READY_OFF command; if he has, the system still comes to command level but the user is not informed.)

A user is initially at command level when the lead task of a user task group is the command processor.

When executing a command function, command level can be returned to in one of two ways.

- o At normal termination of a command function, the system returns to command level and awaits the entry of another command. This command can be some other function that the user desires to execute, or it can be a BYE command, indicating that he has no further work to do and wishes to terminate the current session.
- o The user can interrupt the execution of an invoked command by pressing the "break" or "interrupt" key on his terminal. The system then responds with the break message. At this point he can enter the START command to resume processing where it was interrupted, or he can enter a new command as described in the *Commands* manual.

Functions Performed at Command Level

When a command such as COPY, CWD (change working directory), or EGR (enter group request) is read by the command processor, the system spawns a task whose objective is to fulfill the requirements of the command. This action effectively consists of the following steps:

- o A task is spawned naming the requested bound unit i.e., command name. Task spawning implies task creation, i.e., the allocation and initialization by the system of any control structures and data areas required for task control.
- o The loader is called to load the requested bound unit
- o A request for its execution is placed against the created task and the command processor enters the wait state to await completion of the requested task (command). At this point the system leaves command level, which can be returned to only by completion of execution of the command or by pressing the "break" or "interrupt" key on the terminal, as described previously.
- o If the command is an EGR, it places a group request against an application task group and then the EGR command terminates. The request is queued if there are other outstanding requests against the application task group from previous EGRs.
- o When the command terminates, the spawned task is deleted and a ready message is optionally issued to indicate that the system has returned to command level and can accept further commands.

COMMAND LINE FORMAT

Commands are read and interpreted by the command processor, which executes the lead task in the batch task group, or can execute as the lead task in an online task group. Each command causes a task to be spawned within this task group to perform the requested function (e.g., create a task within an existing group, enter a group request, dump a file). When the execution of a command terminates, control is returned to the command processor, which can then accept another command.

A command line to the processor is a string of up to 127 characters in the form

command-name [arg_1 . . . arg_n] [; command-name [arg_1 . . . arg_n]] . . .

where command-name is the pathname of the bound unit that performs the command's function. Each subsequent arg entry is an argument whose functions are described in the following sections. A command line can span two or more physical lines. A line is concatenated with the next line by ending it with an ampersand (&). A command line consisting of two or more concatenated lines can be cancelled by entering a single ampersand on the next physical line. More than one command may be included in a command line by ending each command with a semicolon.

Arguments

An argument of a command is an individual item of data passed to the task of the named command. Some commands require no arguments; others accept one or more arguments as indicated in the syntax of each command description. The types of arguments used are:

- Positional argument – An argument whose position in the command line indicates to which variable the item of data is applied. The argument can occur in a command line immediately after the command name or as the last argument following the control arguments, as in the LIST NAMES command.
- Control argument – A keyword whose value specifies a command option. A keyword is a fixed-form character string preceded by a hyphen (e.g., -ECL). It can be alone, as in -WAIT, or it can be followed by a value, as in -FROM xx.

Except for -ARG or when the last argument of a command line is a positional argument, keywords of control arguments can be entered in any order in the line, following the initial

SECTION 5
EXECUTION ENVIRONMENT

TASK GROUPS AND TASKS

System control of user applications and system functions is accomplished within the framework of the task group, which consists of a set of related tasks. The simplest case of a task can be considered to be the execution of code produced by one compilation or assembly of a source program (after the code is linked and loaded).

The operating system allows the user to configure a system dedicated to online applications or a combination of online and batch applications. This flexibility of configuration is based on the concept of the task group as the owner of the system resources it requires for execution.

By defining more than one application task group to run concurrently, the user can achieve multiprogramming. He can step through an application in sequence, by causing tasks in the group to be executed one at a time; or he can multitask an application, by causing tasks within the group to be executed concurrently.

Since multiple applications can be loaded in memory at the same time, contending for system resources, an environment must be defined for each application so that it knows the limits of its resources. This defined environment is called a task group, whose domain includes one or more tasks, a memory pool, files, peripherals, and priority levels. By defining the total system environment to consist of more than one task group, the resources are divided up so that more than one application can run concurrently. For Example, memory is divided into memory pools, and task code of a task group is loaded only into that task group's pool and obtains dynamic memory from that pool.

By using the resources of one task group repetitively, an application can be run as a sequence of job or program steps. To achieve this, a task group can be created by a SPAWN GROUP command to use the command processor, whose function is to process system-level commands. Commands can be submitted only in the framework of a task group whose lead task is the command processor. The command processor is activated as the lead task of the task group. One method of sequencing the steps of an application task group is to submit a command to the command processor to read an application command file containing a sequence of names of bound units (files of executable code), where each bound unit corresponds to a program. When a bound unit name is encounted in the file, that bound unit is loaded and executed before the next bound unit name is read. The following is an example of bound unit names in a command file:

```
REP_DATA (The name of a program that gathers report data)
PR_RPT   (The name of a program that prints the report)
```

Another method of sequencing application steps is to issue a SPAWN TASK command for each task to be executed. The SPAWN TASK command causes the task to be loaded, executed, and then deleted. Provided the command processor is instructed to wait for completion of each spawned task, the tasks in the task group can be executed in sequence. For example:

```
ST 1 -EFN REP_DATA -WAIT (Spawn a task to gather report data)
ST 1 -EFN PR_RPT -WAIT (Spawn a task to print the report)
```

Since the Level 6 can be thought of as a set of processors – the central processor, each input/output device, and the real-time clock – the above procedure can be used to attain another effect, that of multitasking within one task group. Consider the situation when the command processor is the lead task, and it reads a file containing SPAWN TASK commands; it does not wait for the execution of the individual tasks, but continues to spawn tasks until it reads an end-of-file or &Q directive. All these spawned tasks are loaded and run

concurrently in this task group, contending among themselves for the resources defined for the task group. For example:

```
ST 1 -EFN REP_DATA (Spawn a task to gather report data)
ST 1 -EFN PR_RPT (Spawn a task to print the report)
```

The command processor does not have to be the lead task of a task group. An application consisting of one task could execute in a task group whose lead task is the application task. Should your application require step control or multitasking, but you do not need the control through commands, you can generate a task group whose lead task contains assembly language system service macro calls whose functions are analogous to the CREATE and SPAWN commands.

The above situations are illustrative and do not exhaust the various ways that you can control program execution.

To summarize: a task group is the owner of system resources, and the context in which system control of tasking is accomplished. A task may be characterized as the *execution* of a sequence of instructions that has a starting point and an ending point, and performs some identifiable function. It is the unit of execution of the operating system, and its execution must be requested through the Monitor software.

The source language from which task code is derived may be any of the languages supported by the operating system. Source code is compiled (or assembled) and linked to form bound units consisting of a root and zero or more overlays.

Application Design Benefits of Task Group Use

Designing an application around the task group provides:

- Intertask communication
- Operating system control of multiple, unrelated task groups

Intertask Communication

The tasks in a task group execute asynchronously in response to the interrupt-driven nature of the operating system and to a linear scan of priority levels assigned to each task group. Tasks communicate through the control structures supplied with each request for task execution.

Asynchronous tasks provide effective software response to information received from real-time external sources such as communications or process control systems. Usually, the task that is activated to handle the interrupt from the external source has a higher priority and a shorter execution time than the task that processes the information. The task that responds to the interrupt will use the operating system to request the execution of the processing task, supplying along with the request the control structure containing a pointer to the new information to be processed. The operating system responds to the request by activating the requested task, or by queuing the request if there are other requests for the execution of this task still pending.

Communications applications would have a high priority task to examine data received at random intervals and decide which processing task should handle the data. This high priority task uses the operating system to queue requests for the processing task, thereby accommodating peak-load conditions in which data is received faster than it can be processed.

In a process control system, the real-time clock might provide the interrupt that causes the high priority task to scan and update temperature, thickness, or raw material level sensors that monitor the physical status of the process. This information would then be passed to a processing task that determines the necessary adjustments based on the new data. Then a third task, having a priority between the other two, could be requested to make whatever changes are required; for example, to change the flow rate of material entering the process by closing a valve.

These two brief examples illustrate the value of priority assignments and communication facilities between tasks.

The Configuration Load Manager (CLM) reads a directive file, and from the specifications supplied, it sets up memory pools and indicates to the loader what system and user-written software is to be resident for the life of the system.

The numbers and characteristics of memory pools are specified in MEMPOOL directives to the CLM. The system and batch pools are defined by an "S" or a "B" on their respective MEMPOOL directives; all other pool definitions are application online pools.

*

Online Pools

Online memory pools are defined in MEMPOOL directives submitted to the CLM during configuration. The definition of online pools requires careful consideration based on the following facts:

- The operating system acquires space on behalf of tasks of a task group from the pool belonging to that task group—not from the system pool (for the exception to this convention see "Sharable Bound Units" later in this section). This means that work space for the task and space for some of the file control and other data structures must be included in the calculation of the task's memory pool size. See the *System Building* manual for these size calculations.
- Online pools can be shared by more than one task group, unless the pool is specified as a serial usage pool.
- The batch pool can be rolled out to accommodate the extension of an online pool into the batch pool area.

There are two types of online pools, exclusive and nonexclusive pools. Online pools can also be specified as expandable and may expand into the batch pool space, thus forcing a rollout of the batch task group. Expandable online pools need not be contiguous with the batch pool.

Calculation of the desired size of system pools is necessary before a trial configuration can be made for an application. However, these calculations may be rough approximations in the early stages of application development. Both the system pool and batch pool (if any) must be explicitly defined in size. The configuration process allows one physical area of memory in the online pool area to be defined as having a size equal to the remaining memory available (* convention) without making precise calculations. The * convention for nonexclusive pools is illustrated below.

*

Exclusive Online Pools

An exclusive pool is one whose boundaries do not overlap those of other pools. An exclusive pool is defined in a MEMPOOL directive to CLM with "E" as the first parameter. The lower part of Figure 5-1 shows a configuration of five online exclusive pools. Each pool is to be used for the tasks of one or more task group. The pools are shown "empty" as they would be at the end of the configuration process.

*

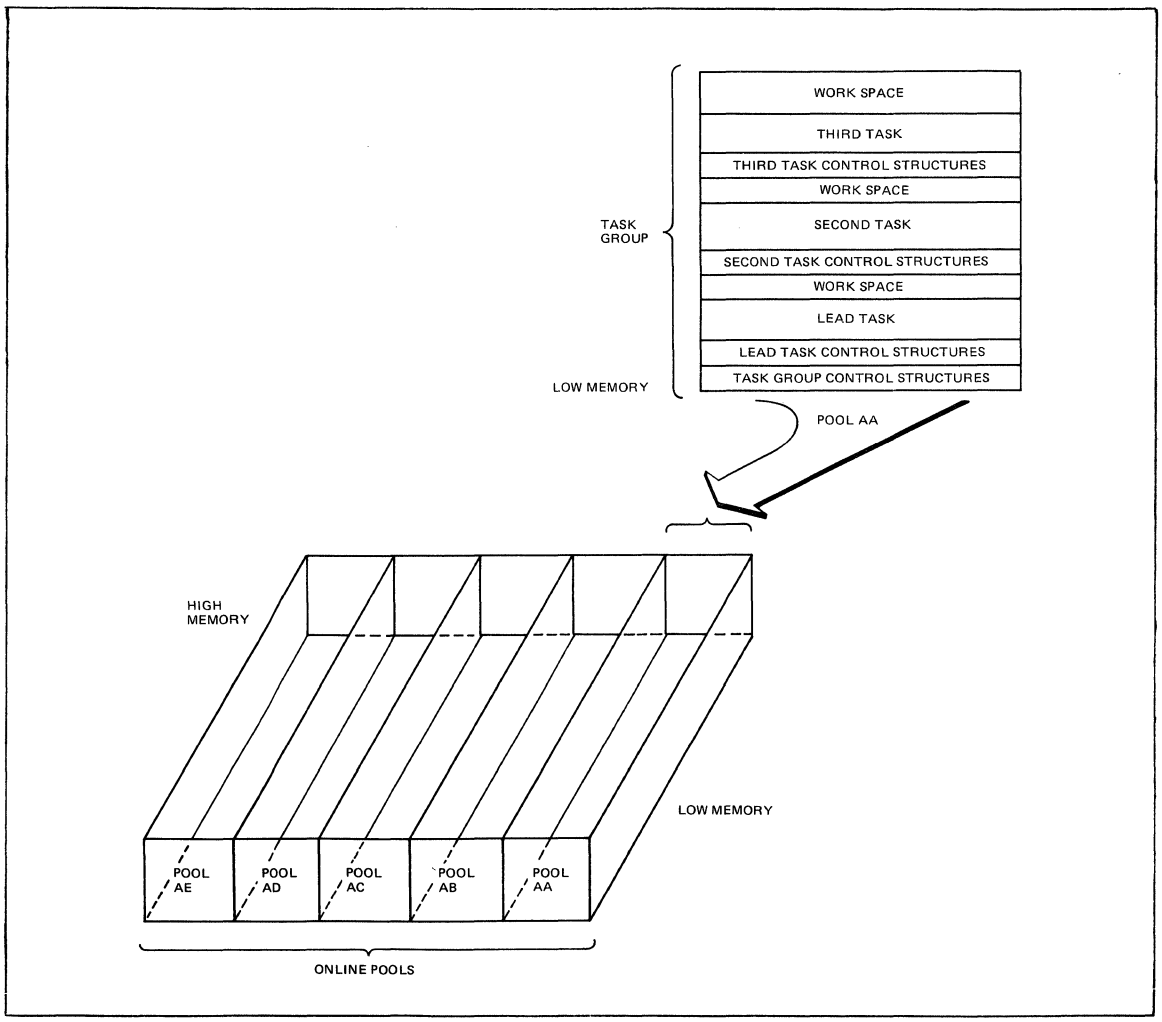

Figure 5-1. Exclusive Memory Pools and Contents

The upper part of the figure is an idealized picture of the contents of pool AA at some instant during processing and shows the memory layout for three concurrent tasks. The figure does not accurately reflect the fact that memory is allocated and returned (in assembly

language programs) dynamically when needed, and work space might not be contiguous to the task code that requests it. Also, memory is allocated in contiguous blocks according to an algorithm that uses multiples of 32 word blocks to calculate the amount of space to assign to a pool. So the "holes" that would normally be present as a result of the operation of the algorithm are missing. The memory manager adds one plus $AF words for control information to the requested amount of space, divides this value by 32, rounds up to the next whole number, and allocates this calculated amount of memory to a task.

The tasks in pool AA (and in all task groups) have priorities assigned to them that are relative to the priority of the lead task, eliminating conflict for resources.

The generation of task groups to use the pools in the figure could be carried out entirely from the operator's terminal or the system START_UP.EC command file using the command processor. Alternatively, after creating a group whose lead task is the command processor, that group could be used to generate task groups and tasks by reading its command file.

The characteristics of exclusive pools:

- o Have "E" as the first parameter on a MEMPOOL directive
- o Consist of one or more sets (all pools are defined by one or more MEMPOOL directives)
- o Can be expandable (can cause roll-out of the batch task group)
- o Have an explicit size except that one exclusive pool may have an * size if no other pool has an * size.

Nonexclusive Online Pools

A nonexclusive pool set is a set of pools whose boundaries overlap those of other nonexclusive pool sets so that some memory locations are common to both pool sets. Figure 5-3 shows two pool sets which are really alternative definitions of the same physical memory area.

Nonexclusive pools are defined in MEMPOOL directives to CLM with a blank first parameter. The characteristics of nonexclusive pools:

- o Have a blank first parameter on a MEMPOOL directive
- o Are an explicit size (except that the last pool in each pool set may be given an * size provided no exclusive pool has an * size.)
- o Can consist of more than one set
- o Can be expandable (can cause roll-out to the batch task group)

Sharing Memory Pools

There are two ways of sharing memory pools. The first method involves assigning two or more task groups to the same pool. As these tasks execute, they contend for the same memory space. Therefore, they should be designed so that they can be suspended or take some alternative action when no additional memory is available.

The second method of allowing task groups to share memory involves the definition of nonexclusive pool sets. Figure 5-2 shows how this might be done.

Pool sets SX/SW/SV and SZ/SY represent alternate definitions of the same physical area of memory. This method of memory use has the advantage of providing flexible and efficient use of resources at any one time. But it has the disadvantage that, unless task group requests are very carefully planned, software deadlock (memory usage conflicts) can occur – the operating system does not prevent it. For example, if task groups using the SZ/SY pool set never execute until those using the SX/SW/SV pool set have terminated, there will never be a problem of deadlock (if only one task group uses each pool).

However, assume that a task group assigned to pool SY is generated and acquires some of the pool space. Then, a task group assigned to pool SW is generated and acquires some space. If each group requested its remaining space, and was willing to wait until the space was available, a deadlock would occur–neither task would ever complete.

The surest way to avoid potential memory usage conflicts is to define all online pools as exclusive pools, and additionally to confine pool use to *one* task group.

If you specify serial usage for a pool, the operating system will not allow a group that uses the pool to be created when another group is already using the pool.

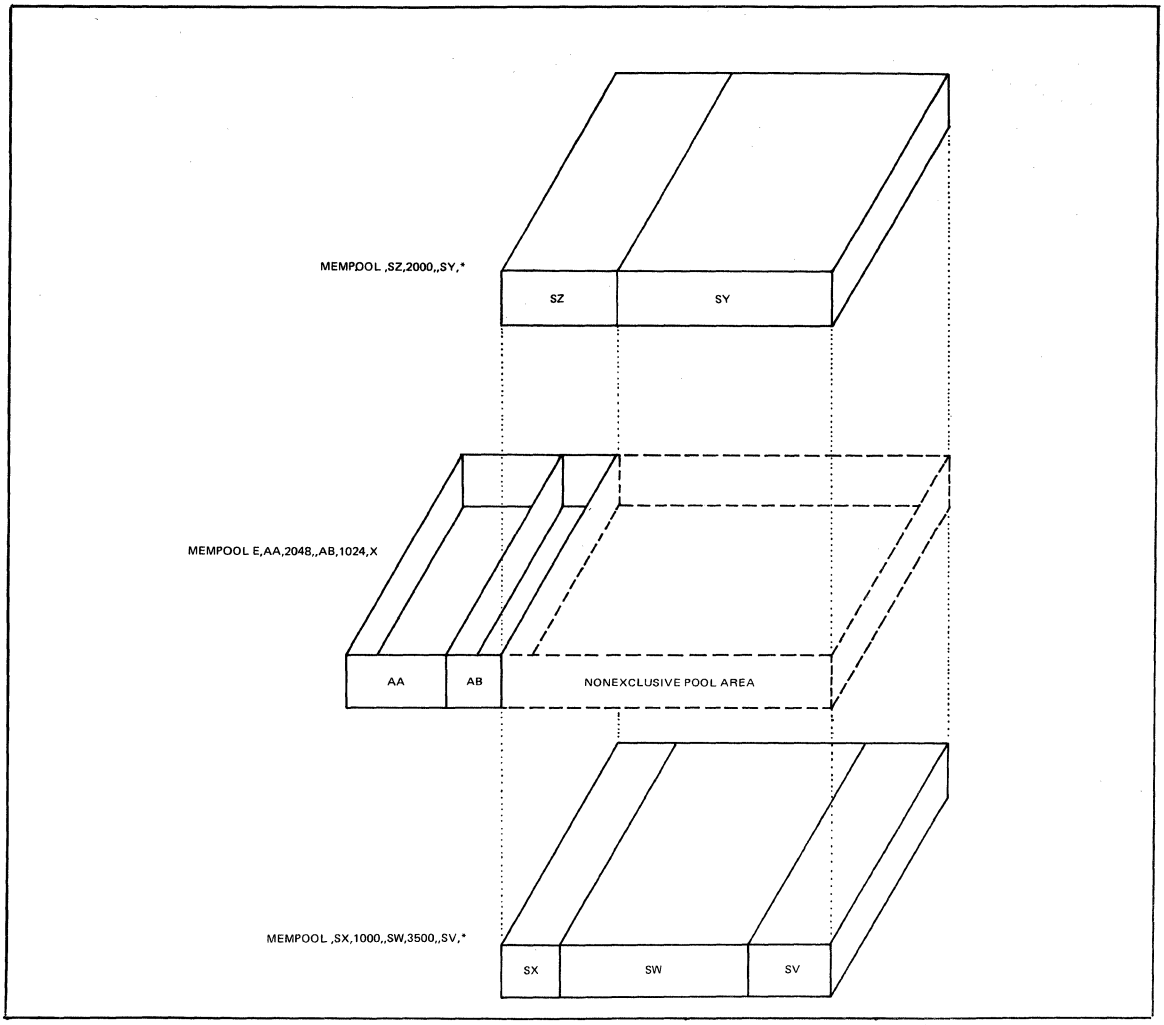

Figure 5-2. Exclusive and Nonexclusive Pool Sets

Batch Pool and Roll-Out

If you configure a system to include a batch pool (by specifying "B" as the first MEMPOOL parameter), it is a resource of the batch task group, whose lead task is the command processor. The usual use for the batch dimension is for program development programs that can be rolled out to provide additional memory for online tasks.

Tasks executing in the batch pool are subject to roll-out when a task executing in an extendable online pool exhausts its assigned memory pool. The operating system initiates roll-out of the batch pool as soon as all batch task group I/O transfers are complete. If no I/O operations are in progress, the task group will be suspended immediately and the entire pool area written out to the roll-out file (>SID>ROLLOUT) on the bootstrap disk as one large record.

Online tasks that use the memory extension capability must be designed so that they can wait for roll-out to occur. Furthermore, unless online task memory requests are coordinated, they could cause "thrashing"—the rapid roll-out/roll-in of the batch pool. Once roll-out is started, it will complete before roll-in occurs, even though the condition that caused roll-out has been removed. For example, Task A requests memory; none is available in its pool; roll-out is requested; Task B immediately releases a block of memory in the same pool large enough to accommodate Task A's request. Roll-out will proceed anyway. Roll-in is initiated when the expanded memory usage is over only if no explicit roll-out request is received.

Operator commands can be used to cause roll-out and roll-in: the SSPB (suspend batch) command causes execution of the batch task group to be terminated temporarily and the group to be rolled out of memory; the ACTB (activate batch) command causes the suspended batch task group to be rolled back into memory and execution to be resumed.

Batch Task Group

The properties of the batch task group and its tasks are as follows:

- The lead task is always the command processor.
- Tasks can only refer to peripheral devices, or mass storage directories and files that are marked shareable.
- Task code cannot execute privileged central processor operation codes (I/O, HALT, LEV).
- Tasks cannot issue system service macro calls or commands that alter the set of task groups defined to the operating system, e.g., CREATE GROUP, DELETE GROUP, ABORT GROUP, SPAWN GROUP.
- A monitor I/O read request will have its boundaries verified if the boundaries fall within the batch pool.
- Real-time application tasks should not be scheduled for execution under the batch task group if the batch pool is subject to roll-out.

Operating System Area

In response to CLM directives, the following software components and data structures will be located in the fixed operating system area after the configuration process is complete:

- Basic operating system software plus resident overlays (RESOLA directive)
- User-written extensions to operating system (LDBU or DRIVER directive)
- Device-drivers
- Intermediate request blocks needed for task groups (SYS directive)
- Trap save areas (SYS directive)
- Overlay area(s) for system software (SYS directive)
- File control structures (file description block (FDB) for nondisk devices)

The operating system area is fixed—its contents remain the same for the life of the system – in contrast to other memory areas whose contents can vary. Almost all code loaded into this area is reentrant so that a single copy of the code is available to multiple users, thus minimizing memory requirements.

System Pool Area

The area adjacent to the resident software area is called the system pool. This area contains the system task group. In addition, the system pool accommodates the following elements:

- Current function invoked by an operator command
- Extended trap save areas (TSAs) needed during processing
- Control structures for the batch task group
- Shareable bound units
- File system directory and file definition blocks

System Task Group

The system task group differs from other task groups in the following ways:

- Cannot be aborted or suspended
- Always has read and write access to all of memory

- o Handles all system dialog (including operator commands) through the designated operator terminal
- o Never terminates, so it cannot be requested

Batch Task Group Control Structures

The following control structures are found in the system pool area whenever a batch memory pool is configured:

- o Group control block (GCB)
- o Logical resource table (LRT)
- o Logical file table (LFT)
- o Task control block (TCB)
- o Batch request block

File Control Structures in the System Pool Area

The elements in the system pool area that are used for file control consist of:

- o File description block (FDB)
- o All buffer for shareable files
- o Buffers for shareable files

Pool Attributes

The user can exercise more control over memory usage by providing online memory pools with specialized attributes, as described below.

Protected Memory Pools[1]

A memory pool may be "protected" if it is so specified at configuration (by a CLM command). A protected pool is one into which a task running in another pool may not write. Through use of the Memory Management Unit (MMU), the operating system will prevent write intrusion by foreign tasks. Such a task will receive an error notice from the operating system when an intrusion is attempted.

The special size constraints that apply to protected pools are described in the System Building manual.

Contained Memory Pools[1]

Any memory pool but the system memory pool may be "contained" if it is so specified at configuration. Tasks running in a contained pool are prevented from writing outside their own pool area. The constraints that apply to the size of contained memory pools are the same as those that apply to protected memory pools.

Unprivileged Memory Pools

At configuration, any memory pool except the system pool or the batch pool may be declared "unprivileged". A task running in an unprivileged pool cannot execute privileged instructions, and will trap if such an execution is attempted.

The system pool is always privileged and the attribute cannot be altered.

The batch pool is always unprivileged and the attribute cannot be altered.

Exclusive and nonexclusive pools are privileged unless specified to be unprivileged

Serial-Usage Memory Pools

An exclusive or nonexclusive memory pool may be declared serial-usage. If so declared, such a pool may be used by only one task group at a time.

Multi-Pool Memory Protection

On a system having a Memory Management Unit (MMU), a user may specify the write protection/containment that the system is to provde. At CLM time, the user selects one of the following options. The option selected prevails until the sytem is reconfigured.

[1] Applies only to configurations having a Memory Management Unit (MMU).

1. No protection or containment; i.e., no utilization of the MMU.
2. Protection of the system memory pool and/or containment of the batch memory pool.
3. Protection and/or containment of selected memory pools in addition to those of option 2.

Note that protection applies to memory pools and not to task groups. Thus, groups sharing a memory pool are not protected from each other. Nor is the only group in a memory pool secure from intrusion if the pool is a nonexclusive pool.

If a task group is to be protected from all other groups, it must be the only group using an exclusive memory pool, and option 3 software protection must be specified at system configuration.

Memory Layout

To obtain efficient use of memory and of the Memory Management Unit the Configuration Load Manager (CLM) sorts the memory pools in a configuration as follows:

- o The system pool is in the first available memory after the system date structures.
- o Nonprotected, noncontained pools are next in order of size; the smallest one comes first.
- o Protected and/or contained pools come last in order of size; the smallest one comes first.
- o All nonexclusive pools are considered to be a single pool, for purposes of memory allocation.

Selecting Memory Pool Attributes For Task Group Execution

The different type memory pools provide system users with the means to respond to the unique demands of multiple application programs. Through the use of memory pools the user can at once exercise control over memory usage and at the same time provide individual task groups with specialized protection attributes.

The degree to which the system can efficiently and effectively handle the concurrent execution of multiple task groups depends on the number and type of memory pools available for use.

Cases 1, 2, and 3, below examine the considerations involved in the selection and use of memory pools.

Case 1:

The user's program consists of a real-time data application program. The program must co-exist with other user applications.

The data application program accepts data based on unpredictable external stimuli. The application permits a "saturation" effect to occur when data collection exceeds the effective rate of processing. The occurrence of "saturation" represents a signal to the application to initiate a data selection strategy.

The user should select an on-line pool of sufficient memory size to control the maximum desired amount of data allowed to accumulate.

Case 2:

The user's application consists of process control software with a periodic need for all of available user memory.

The application requires the total memory of the system only 30 seconds of every five minutes. For the remainder of the time, only 10% of user available memory is required.

In this case the process control application ought to be loaded into an extendable online pool. The size of the online pool should be 10% of available memory. The applications periodic need for all of available memory can be fulfilled by defining a batch pool that consists of 90% of available memory. Whenever the process control application requires more memory than that in its online pool, then the system will initiate a rollout of the batch pool. During the period when the process control application does not require all of available memory, then the use of the batch pool can be obtained by other applications.

Case 3:

Multiple applications with strict integrity requirements are to coexist with programs under development test.

The user can choose to load each of the applications with integrity requirements into a memory pool that has been designated as "protected". This will insure that the programs under test do not accidently modify data in the programs to be protected.

BOUND UNITS

Task code is derived from the source language of programs that are compiled or assembled to form object units. One or more object units are linked to form a bound unit that is placed on a file. The bound unit is an executable program that can be loaded into memory. A task represents the execution of a bound unit. Each bound unit consists of a root segment and any related overlay segments.

Overlays

To minimize the amount of memory required to execute a bound unit containing application code, the bound unit can be created as a root and one or more overlays. Object units whose code is to be loaded as overlays are defined as overlays by the Linker. The use of overlays requires careful planning so that required code is not lost or repetitively loaded.

Nonfloatable and Floatable Overlays

Overlays are part of a bound unit which comprises a root or a root and one or more overlays. There are two types of overlays: the nonfloatable overlay that is loaded into the same memory location relative to the root each time it is requested, and the floatable overlay that is linked at relative location 0 and can be loaded into any available memory location.

Floatable overlays must have the following characteristics:

- o External location definitions in the overlay are not referred to by the root or any other overlay.
- o The overlay makes no immediate memory addressing (IMA) references to itself, and no displacement references to the root or any other overlay.
- o The overlay can contain IMA references with or without offsets to the root or any other nonfloatable overlay.
- o The overlay does not contain external references that are not resolved by the Linker.
- o The overlay must be linked *after* all nonfloatable overlays have been linked.

A user program can use one or more areas of its available memory for placement of floatable overlays. To be most effective, the program must perform its own memory management of these areas. To perform memory management, a user-written assembly language overlay manager must be linked with the root of the program bound unit. Adequate memory space must also be provided in the pool of the requesting task. If the user program does not control the placement of a floatable overlay, the system will place the overlay in available space in available memory.

Assembly language programs can use system service macro calls to load and execute nonfloatable overlays; memory management is handled by the Monitor. Similarly, COBOL programs can use CALL/CANCEL statements to control nonfloatable overlays. FORTRAN and RPG programs must link a user-written assembly language overlay manager with the application program.

Resolving References

Forward references can be made to symbols defined in object units to be linked later. Backward references can be made to symbols previously defined provided that the defined symbols were not purged from the Linker symbol table by a Linker BASE or PURGE directive. Since the specification of the BASE directive removes from the Linker symbol table all previously defined, unprotected symbols that are at locations equal to or greater than the location designated in the BASE directive, you must either define all symbols in a nonoverlaid part of the root or plan the linking of subsequent overlays so that purging of needed symbols does not occur.

Floatable overlays can refer to fixed addresses in the root or nonfloatable overlay, but cannot refer to addresses in another floatable overlay.

When a root or an overlay of a bound unit is loaded, the loader examines the attribute tables associated with the bound unit, if an alternate entry point is specified. The loader tries to resolve any references to symbols that remain unresolved at load time by searching the system symbol table (i.e., the resident bound unit attribute table); it cannot resolve any references to symbols that do not exist in that table (Linker symbol tables do not exist at load time).

Sample Overlay Layout

Figure 5-3 illustrates the layout of overlays in a memory pool AA. The Linker directives to create and specify the location of these overlays are described in the *Program Execution and Checkout* manual.

When the root is loaded, the largest contiguous amount of memory necessary to accommodate the *root* and all *nonfloatable* overlays is allocated. Except for space for any floatable overlays, no other memory requests need be made. In the figure, this memory area begins at relative 0 of the root, and continues to the end of object unit OBJD. The root consists of object units OBJ1 and OBJ2. When loaded, OBJ5 of overlay ABLE will replace the previously loaded OBJ2 code of the root. Similarly, the overlay locations were specified so that OBJC of overlay ZEBRA will replace part of OBJB.

Shareable Bound Units

Using shareable bound units is a way of minimizing application task group memory requirements while making reentrant code available to multiple tasks. Unlike permanently resident bound units that are loaded during system configuration, shareable bound units are transient in the system pool and are loaded during processing. A counter is incremented each time a request is made for the bound unit, and the unit remains in memory as long as a task is using the code. As soon as the counter is decremented to zero, the system pool space occupied by the bound unit is returned to available status.

Operator commands can be used to load and then unload a shareable bound unit.

To be recognized as shareable by the loader and loaded into the system pool, the bound unit must have been so marked by the Linker in response to a SHARE directive when the bound unit was linked. If the system pool does not have enough space to accommodate the bound unit, at a given instant, it will be placed in the pool associated with the task group that requested the bound unit, and cannot be shared.

Shareable bound units and the operating system extensions that are loaded when the system is configured differ in another way. Namely, operating system extensions can be referred to directly by any task, but a shareable bound unit must be accessed as a task. The reason for the difference is that an operating system extension is loaded when the system is configured–its symbols are included in the system symbol table at that time. Since there is no symbol definition once configuration is complete, and since a shareable bound unit is loaded *after* the system has been configured–no entry for it exists in the system symbol table, and it must be accessed as a task.

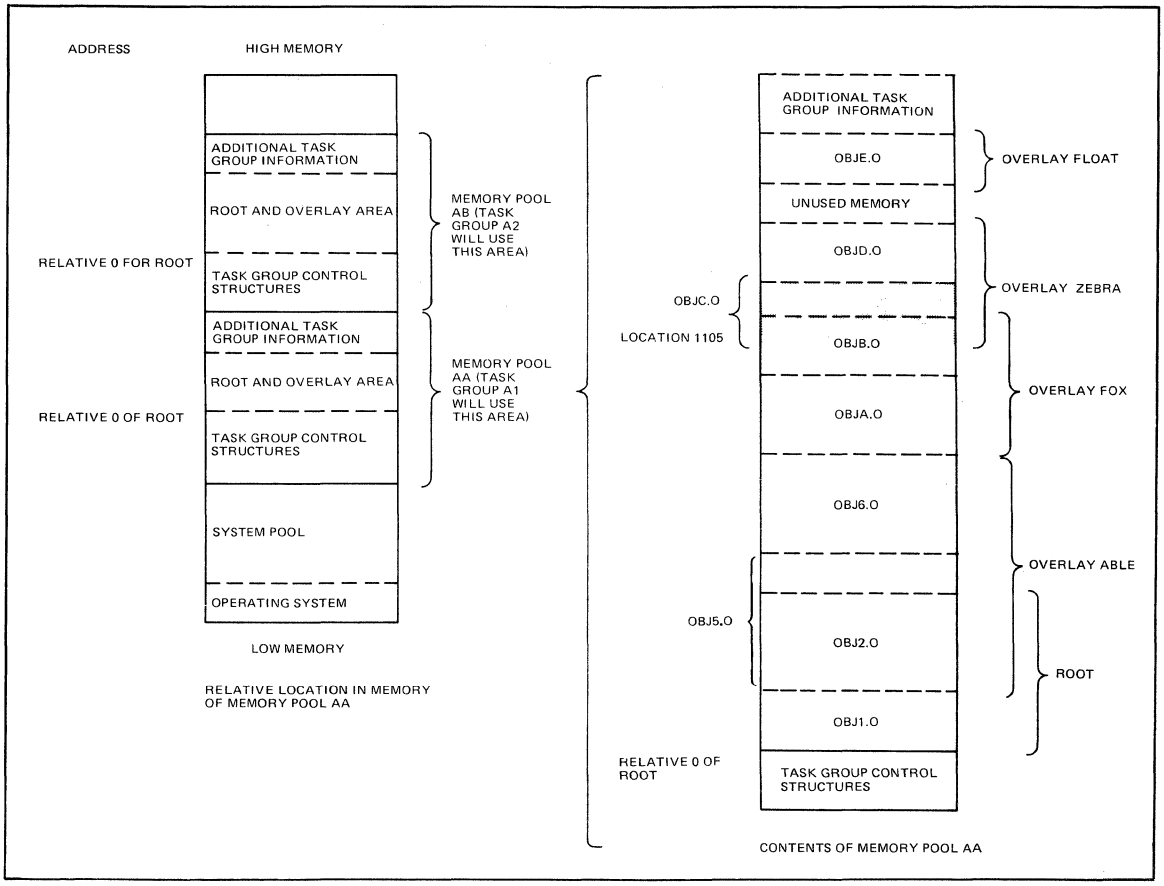

Figure 5-3. Overlays in Memory

Table 5-2 compares permanently resident operating system extensions and transient shareable bound units.

TABLE 5-2. COMPARISON OF OPERATING SYSTEM EXTENSIONS AND SHAREABLE BOUND UNITS

Characteristics	Operating System Extension	Shareable Bound Units
Multiple Users	Yes	Yes
Permanent Resident (fixed area)	Yes	No
Temporary Resident (dynamic area)	No	Yes
Symbols in System Table	Yes	No
Accessed symbolically?	Yes	No
Accessed as a task?	Yes[a]	Yes
Can have overlays?	No	Yes
Called by Bound Unit Name	No[b]	Yes

[a] If the extension is an assembly language bound unit, it may have within it sections of code or control structures controlled by semaphores which would be accessible to other assembly language tasks.

[b] The operating system does not "remember" extensions by their names; a request for one by name results in another copy being brought into memory.

Loading Bound Units (Search Rules)

The loader follows preestablished search rules when searching through a set of directories to locate a bound unit to be loaded. The loader initiates the search in response to a command which contains an argument naming the bound unit to be loaded.

Assigning Priorities to Application Tasks

Priorities are assigned to user task groups and tasks when they are created or spawned. The command to generate a task group contains an argument that represents the base priority level for the task group. The base priority level is relative to the highest system physical priority level, i.e., the highest interrupt priority level number used by the system in that configuration. When a task group is assigned a base priority level of zero, the lead task group executes at the physical interrupt priority level that is the next level above that of the highest level system task. When other tasks in the same task group are created or spawned, they are given level numbers relative to the base priority level assigned to the task group. The physical interrupt priority level at which a task executes is the sum of the highest system physical priority level plus one, the base priority level of the task group, and the relative priority level of a task within that group. This total must not exceed 62.

User tasks that are to execute in the online dimension are usually given higher priorities (lower level numbers) than those in the batch dimension. Tasks that are I/O bound should be run at a higher priority than tasks that are central processor bound. This permits I/O-bound tasks, which run in short bursts, to issue I/O data transfer orders as needed, wait for I/O completion, and, while in the wait state, relinquish control of the central processor to the central processor-bound tasks. Otherwise, if the central processor-bound tasks have a higher priority, the I/O devices would be idle while I/O bound tasks waited to receive central processor time.

Logical Resource Number (LRN)

An LRN is an internal identifier used to refer to task code and devices independently of their physical priority levels. Use of LRNs makes assembly language application task code independent of priority levels so that if circumstances require a change in priority levels, the task code does not have to be reassembled.

Device LRNs

LRNs are assigned to devices in the CLM DEVICE directives when the system is configured. The operator's terminal is assigned to LRN 0 at configuration. The bootstrap device is given an LRN value of 1, and, if an MLCP-connected operator's terminal is configured, the system assigns it an LRN of 2. Figure 6-3 is an example of priority level assignments for devices and system tasks and the related device LRNs.

LRN	LEVEL	
	0	
	3	INT
	4	CLOCK
0	5	OPERATOR'S TERMINAL
1	6	DISK
3	7	LINE PRINTER
4	8	SERIAL PRINTER
5	9	CARD READER
	10	OPERATOR INTERFACE MANAGER INTERRUPT
	11	SYSTEM TASK

Figure 6-3. Example of LRN and Priority Level Assignments to System Tasks and Devices

Application Task LRNs

LRN assignments to application program tasks are not dependent on the system configuration on which the application task group is running. LRNs are assigned to task code within an assembly language application program through specification of the create group/task macro calls, as well as the macro calls that build data structures ($IORB, $TRB, etc.). LRNs can be assigned at the control language level through the use of the commands (including operator commands) for creation of tasks groups and tasks. An LRN for an application task can have any value from 0 through 252. Within a task group, the LRN for each task must be unique. More than one LRN can be associated with the same level. For example, two tasks at level 23 can be assigned LRNs of 28 and 29, respectively.

Logical File Number (LFN)

Logical file numbers are internal file identifiers that are associated with file pathnames either at the assembly language level, or for high-level languages at the command level, through GET or ASSOCIATE commands.

Inter/Intra Task Group Communication

Whether or not information can be passed between task groups and tasks depends upon the following considerations:

- o Language in which task code is written
- o Use of the same file by more than one task group
- o Use of the message facility

Language Considerations

Task code written in assembly language can pass information to other assembly language tasks in the same task group by using variable-length request blocks. (See the *System Service Macro Calls* manual for details about building these data structures.) High-level languages cannot use this mechanism directly, but would require called subroutines written in assembly language.

Use of Common Files

Tasks within the same (or different) task group can communicate via disk files. The concurrrency status must be the same for all tasks using the files. The requesting tasks must have access rights to the files.

Use of the Message Facility

The message facility allows online communications between two task groups using assembly language code. The task groups communicate by sending and receiving messages to/from message queue container called mailboxes.

In using the message facility, the task groups issue commands to prepare the mailboxes and system service macro calls to send and receive the messages.

Mailbox Preparation

The user or operator must create the mailbox root directory on the local volume, create the needed mailboxes, and set access controls on the mailboxes.

The mailbox root directory is the directory that is to contain the simple names of the mailboxes. It is created through a standard CREATE DIRECTORY command.

The user creates each needed mailbox through a CREATE MAILBOX command. This command creates a directory corresponding to the mailbox name and a file ($MBX) within that directory defining the mailbox attributes.

Since memory queuing of the messages is required, the CREATE MAILBOX command must have the -MEM argument; the queue capacity (in bytes) is specified by the -SIZE argument.

To prevent unauthorized use of the message queues, the user should set access controls on each created mailbox. Access rights must be set as follows:

- o The sender must have list access on the directory defining the mailbox.
- o The receiver must have read access on the $MBX file for a given mailbox.

The following is an example of mailbox preparation (see the *Commands* manual for details on the various commands).

1. Create the default mailbox root directory.[1]
   CD >MDD
2. Create a mailbox (directory and file).
   CMBX > SMITH -MEM -SIZE 100

[1] If the user creates a mailbox root directory named MDD, then the format of the CMBX command can include a simple pathname (the system assumes the absolute pathname to be >MDD>mailbox directory>$MBX). If the user creates a mailbox root directory not named MDD, then the format of the the CMBX command must include an absolute patname.

3. Create another mailbox (directory and file).
   CMBX >JONES -MEM -SIZE 50
4. Set read access on the file created in step 2.
   SET_ACL >MDD>SMITH>$MBX R *.Smith.*
5. Set list access on the directory created in step 2.
   SET_ACL>MDD>SMITH L *.*.*
6. Set read access on the file created in step 3.
   SET_ACL>MDD>JONES>$MBX R *.JONES.*
7. Set list access on the directory created in step 3.
   SET_ACL>MDD>JONES L *.*.*

Sending and Receiving Messages Between Task Groups

A task group that wishes to send a message to another task group must issue an Initiate Message Group ($MINIT) macro to open the send portion of the message facility. The task group then issues Send ($MSEND) macro calls to send message data. In the same way, a task group that wishes to receive a message from another task group must issue an Accept Message Group ($MACPT) macro call to open the receive portion of the message facility. The task group then issues Receive ($MRECV) macro calls to receive message data. When the message is received, both the sending and receiving task groups should issue a Terminate Message Group ($MTMG) macro call.

The message facility can be used most effectively if it remains active for as long as the two task groups wish to communicate. To accomplish this, the task groups should issue the Initiate Message Group and Accept Message Group macro calls to invoke the message facility. Then a task group can issue Send macro calls whenever it has data to send; the task group that is to receive the data can issue Receive macro calls when it wants to receive the data. When the message (one message with multiple records) has been received, both task groups should issue a Terminate Message Group macro call.

Refer to the *System Service Macro Calls* manual for a full description of the message facility macro calls.

Sending Messages To A Task Group

A task group wishing to send a message to another task group issues an Initiate Message Group ($MINIT) macro call to inform the system that a message is to be send and to provide the name of the queue (mailbox) to which it will be sent. The task group than issues one or more Send ($MSEND) macro calls to send message data. The Send macro calls name the message is returned by the Initiate Message Group macro call.

The message is made up of records. The last record carries the end-of-message indicator; intermediate records may carry an end-of-quarantine indicator. The message is terminated by the Terminate Message Group ($MTMG) macro call or, alternatively, by the SEND ($MSEND) macro call.

The sending task group can issue the Count Message Group ($MCMG) macro call to ascertain the number of messages currently in the mailbox.

Receiving Messages From A Task Group

A Task group wishing to receive a message from another task group issues an Accept Message Group ($MACPT) macro call to inform the system that a message is to be received from a named queue (mailbox). The task group then issues one or more Receive ($MRECV) macro calls to receive message data. The Receive macro calls name the message id received by the Accept Message Group macro call.

The receiving task group can request the message in record sizes that are other than the sizes in which the message was sent; the receiving task group delimits the amount of received data by range, end-of-quarantine-unit, or end-of-message specifications. The task group must issue a Terminate Message Group ($MTMG) macro call when the entire message has been received.

The receiving task group can issue a Count Message Group ($MCMG) macro call to ascertain the number of messages in the mailbox.

Task and Resource Coordination

Tasks can be coordinated in either of two ways:

- o Through the use of tasking requests
- o Through the use of semaphores

Task Requests

One task can request another to execute asynchronously with it, or the requesting task can later wait for the completion of the requested task. Both tasks have access to the request block provided by the requesting task, and thus can pass arguments between them.

Semaphores

Semaphores support an application-designed agreement among tasks to coordinate the use of a resource such as task code or a file. A semaphore is defined by a task within a task group, and is available only to the tasks within that group.

For each resource to be controlled, a semaphore is defined, and given a two-character ASCII semaphore name. This name is a system symbol recognized by the Monitor, and *not* a program symbol that needs Linker resolution. The agreement is: each requestor of a resource whose use must be coordinated issues appropriate Monitor calls to the named semaphore to request or to release the resource. The task that defines the semaphore assigns the semaphore's initial value. The Monitor maintains its current value to coordinate requestors of the resource being controlled. A requestor obtains use of a resource if the semaphore value is greater than zero at the time of the request. A requestor is either suspended waiting for the resource or notified that no resource is available if the value is zero or negative.

Monitor service macro calls are used to:

- o Define a semaphore and give an initial value ($DFSM)
- o Reserve a semaphore-controlled resource ($RSVSM); this macro call subtracts a resource, or queues waiter for the resource; i.e., it *decrements* the current-value counter.
- o Release a semaphore-controlled resource ($RLSM); this macro call adds a resource, or activates the first waiter on the semaphore queue; i.e., it *increments* the current-value counter.
- o Request the reservation of a semaphore-controlled resource ($RQSM); this macro call queues a request block (SRB) if the resource is not available. This macro call *decrements* the current-value counter.

A semaphore is a gating mechanism, and the initial value given to it depends upon the type of control you want to exercise.

SECTION 7
DISTRIBUTED SYSTEMS FACILITIES

The GCOS system is designed to interface with communications and networking products to provide components for implementation of a distributed systems environment. This section describes the following GCOS facilities that use communications software to perform data transfers: the Remote Batch Facility, the Data Entry Facility, utility programs that support file transmission between the Level 6 and other processors, the Terminal Concentration Facility, Level 66 Host Resident Facility, and IBM Workstation software.

REMOTE BATCH FACILITY (RBF)

The Level 6 Remote Batch Facility (RBF) is a software package enabling Level 6 hardware to be used in a remote batch processing environment with Level 66 and Series 6000 host processing systems. Remotely located Level 6 peripheral devices can enter jobs into and receive output from one to four host processors.

The Remote Batch Facility works in conjunction with a host processor and a Front-End Network Processor (FNP), operating under control of General Remote Terminal Supervisor (GRTS) or Network Processing Supervisor (NPS) software.

The Remote Batch Facility can use either of two line protocol conventions that control the flow of data between the Level 6 and the FNP:

- o Remote Computer Interface (RCI)
- o High-Level Data Link Control (HDLC)

The Remote Batch Facility operates under control of the GCOS 6 operating system. Remote batch and GCOS 6 local processing functions that are independent of the host processor can be performed concurrently, provided adequate resources (i .e., memory, peripheral devices) are available. Local processing of the following types can be performed:

- o Program/system development and maintenance
- o User-written processing applications
- o User-written data communications applications

Remote batch terminal (RBT) software is run as a task executing in a unique task group and using the resources reserved for that task group. In systems with adequate resources, the Remote Batch Facility can support the concurrent operation of up to four remote batch terminals. Each RBT permits the batch entry of remote jobs destined for processing in a host system and the receipt of output from those jobs. Each RBT is associated with a stream; i.e., a logical connection that lets data travel between two end points between a device or file at an RBT and a host processor).

In remote batch processing, the following functions can be performed at an RBT:

- o Enter a job or group of jobs for processing by a host processor.
- o Combine input from more than one input medium or file into one job.
- o Obtain the status of jobs in the host processor.
- o Use the transparent binary feature to process cards in nonstandard binary format without performing checksumming.
- o Spool data to a file for temporary storage. Input jobs can be stored on a spool file and later be transferred to the host. Batch output can be spooled and then printed or punched at a later time.

- o Direct that job output be sent to any of the terminal's peripheral devices capable of receiving output, to another RBT, or to the host computer site.
- o Backspace an output file and resume output processing from that point.
- o Change printer forms as specified on GCOS control cards.
- o Abort a file that is being retrieved (output from host) or a job that is being entered (input to host).
- o Restart job output processing, specifying the page and/or line number for printer output or the card number for punch output.

RBF Configuration

The user must configure the system using configuration directives (see the *System Building* manual). He then creates a task group for each RBT, defining initial input and output file assignments, modifying external switches associated with the task group and, finally, invoking the RBT and identifying its processing stream.

Remote Batch Operations

Remote batch operations are controlled by entering commands from either the Level 6 operator terminal or an RBT console. Operator messages are issued on the console and define conditions that may require operator action. If necessary, operator responses can be entered through the console.

The Remote Batch Facility supports multiple communications lines to one to four host processors. The lines can be either all dedicated (which are always connected), all switched (which are connected by dialing a telephone and disconnected at the end of each session), or a mixture of dedicated and switched.

Jobs to be processed can be prepared on cards, and read directly through the card reader by an RBT. Alternatively, the job deck can be prepared using the Editor and read from a file in ASCII code. Or, the cards can be spooled to a file and read in GBCD code.

Output records are delivered to a specified output file or device. If the requested device is in use or inoperable, you can wait for device availability or direct the output to a different device.

Refer to the *Remote Batch Facility User's Guide* for a complete description of the Remote Batch Facility.

DATA ENTRY FACILITY (DEF)

The GCOS 6 Data Entry Facility (DEF) is a multifunctional data entry system. Data can be entered through an operator control station, validated, edited, verified, and communicated to a host computer for further processing.

DEF is a disk system combining Level 6 hardware with DEF software. DEF operates under a task group under GCOS software and can operate simultaneously with other system functions and tasks. It supports up to 12 operator display stations (VIP 7200 terminals) and up to six line or serial printers.

The following functions can be performed using DEF:

- o Develop forms; create, modify, delete, print, and view forms
- o Develop tables; create, modify, delete, print, and view tables. The tables are used to ensure that data is entered correctly or to replace data with other specified data.
- o Enter or modify data records by using a form as a template. The entire form is displayed on the screen, and variable (unprotected) areas can be written into or altered.
- o Verify contents of all or specified fields by rekeying the data.
- o Print entire files or selected records from files. The printout can be formatted or unformatted (all data in a single record is run together on one or more lines).

DEF functions may be run concurrently with other GCOS 6 facilities such as file transmission to and from a host computer.

Interface with Programs

Data entry subroutines can be created to include capabilities such as arithmetic functions (e.g., batch totals) or additional data validation and editing features. The subroutines are automatically executed when the operator reaches the fields where they were specified. Application programs may be run concurrently with other DEF functions; application programs can be used for functions such as printing reports received on a DEF disk from a host system or sorting and merging data files. Data entry subroutines and non-data-entry application programs can be written in COBOL or assembly language.

DEF Operations

All system operations, including data entry and system control, are performed from the VIP 7200 operator display station. Data entry and control is initiated from the keyboard. User-generated information (e.g., data entry forms) and system-generated information (e.g., selection lists and prompting messages), and all entered data are displayed on the screen. Selection lists indicate which options may be selected (e.g., data entry, supervisory functions). Prompting messages request entry of specific responses or information.

Data is entered at an operator display station onto a form created for the particular application. As data is entered, specified data validation and editing takes place. If an error exists, an alarm sounds. Error messages are displayed on the bottom line of the operator display station. The appropriate correction can be made and, if desired, additional data entered. The data entered is stored on disk and is available to be processed, viewed, modified, deleted, printed, or transmitted using the GCOS 6 file transmission utility program.

DEF Supervisory Functions

DEF supervisory functions have the capabilities listed below; access to these functions is protected via a password to prevent unauthorized or accidental access to the functions:

- Copy forms, tables, and data files
- Rename forms, tables, and data files
- Delete forms, tables, and data files
- Assign any number of operator display stations to a selected printer or change the current disk volume assignments for forms, tables, and data files.
- Examine and change the system status
- Change the supervisory password

DEF Utilities

Utility routines provide the following capabilities:

- Display (1) form names, (2) names and types of tables (i.e., extract or verify table), or (3) file names, number of records in each file, and each file's verification status
- Print (1) form names, (2) names and types of tables, or (3) file names, number of records in each file, and each file's verification status
- Display names of disk volumes to which the operator display station has access for reading and writing forms, tables, and data files.
- Temporarily change disk volumes that forms, tables, and data entry files are read from or written to for the operator display station at which the user is working.

DEF Configuration

The user must configure the system using configuration directives (see the *System Building* manual). He must link the DEF object modules to form a bound unit. He then creates a task group for DEF and creates DEF-specific tasks.

For a complete description of the Data Entry Facility, refer to the *Data Entry Facility User's Guide* manual.

FILE TRANSMISSION BETWEEN LEVEL 6 AND OTHER COMPUTERS

File transmission between the Level 6 and a variety of other processors (Level 6, 62, 64 and 66, Series 200/2000, and non-Honeywell processors) is implemented through three utility pro-

grams: TRAN, TRANH, and TRANB. Each of these utility programs permits files to be transmitted to or received from one or more remotely located processors. Each processor must incorporate appropriate file transmission software.

The TRAN utility program provides for file transmission between the Level 6 and one or more Level 66 host processors. The TRANH utility program is used for file transmission between the Level 6 and other Level 6 processors, or between the Level 6 and Level 62, Level 64, or Series 2000 host processors. Both TRAN and TRANH transmit files in ASCII format, using the polled VIP protocol.

A third utility program, TRANB, enables file transmission between the Level 6 and non-Honeywell processors that use the BSC2780/3780 protocol; TRANB converts ASCII data in Level 6 files into EBCDIC 80-character records for transmission, and converts the received EBCDIC records into ASCII format.

Each file transmission program is invoked by a command (either entered on a terminal or included within a user EC command file). The command name corresponds to the name of the utility program invoked: TRAN, TRANH, or TRANB. Each program provides error analysis. For TRAN and TRANH, an initiate/accept dialog between file transmission software in each of the two processors determines whether a file can be transferred. A restart capability is available when transmission between two Level 6 processors or between a Level 6 and a Level 66 processor is aborted due to failure in the transmission line. File transfer can be restarted at any record in the file being transferred at the time of failure.

Multiple file transmissions between the Level 6 and one or more processors can occur concurrently. (For example, the Level 6 could transmit files to a Level 64 and a Level 66 host processor concurrently.) Each file transfer takes place over a different communications line. An argument in the command that invokes the file transmission program specifies whether a specific communications line is to remain connected after a file transfer or is to be disconnected. As long as the line is connected, file transfers can be made by issuing the appropriate command (TRAN, TRANH, or TRANB) for each transfer.

For details on the use of the file transmission utility programs to transmit files to a specific processor, refer to the appropriate file transmission manual.

TERMINAL CONCENTRATION FACILITY

Terminal Concentration permits various types of synchronous and asynchronous terminals to concurrently connect to a Level 6 and have their message traffic concentrated (multiplexed) over one or more links to a host processor. It therefore provides the means to smooth the sporadic traffic patterns associated with terminals into a steady flow on a higher quality line. Concurrently, it reduces the total number of modems and long-distance lines required by multi-terminal applications and improves total reliability. Thus, by coordinating the movement of information to and from a host computer site, TCF relieves the host site of many terminal communications processing operations.

The following are some of the features supported by TCF:

- o Multiple links to the same or multiple different hosts
- o User log-in capability to pick up host type and/or specific host link
- o All terminals concentrated appear to the host as VIP 7700-like terminals
- o Handling of terminal polling functions
- o Concertration of messages
- o Error detection and notification

The Terminal Concentration Facility is designed to run as a standard GCOS6 application. The concentration software does not require exclusive control of the processor, and thus allows other Level 6 functions (GCOS6 user applications, remote batch, data entry, etc.) to share a link to a host processor.

TCF supports the following teletype compatible terminals:

- o TTY 33, 35
- o TN 300
- o TWU 1001 (SARA 20)
- o TWU 1003 (ROSY 24)
- o VIP 7100 ("glass teletype")

The following VIP keyboard/display terminals are supported:

- o VIP 7700R
- o VIP 7700
- o VIP 7760

IBM WORKSTATION FACILITIES

Honeywell provides two software packages for use in communicating with an IBM 360/370 host system: IBM 2780/3780 Workstation Facility and the HASP Workstation Facility. The software simulates the transmission facilities provided by the named workstation; with the workstation facilities, connection can be established between a Level 6 and an IBM 360/370 host system.

2780/3780 Workstation Facility Capabilities

The IBM 2780/3780 Workstation Facility is used for either transmitting batch input to an IBM 360/370 host system or receiving batch output from an IBM 360/370 host system. The following capabilities are provided.

- o Line printer horizontal format control
- o Automatic restart
- o Dual communications interface
- o Auto answer (dial-up operation only)
- o Multiple record transmission
- o Error reporting and retransmission

In addition to the previously specified capabilities, the 3780 operational mode provides the following additional capabilities.

- o Space compression/expansion
- o Conversion Mode
- o Automatic disconnect

HASP WORKSTATION FACILITY CAPABILITIES

The HASP Workstation Facility is used for either transmitting batch input to an IBM 360/370 host system or receiving batch output from an IBM 360/370 host system. The following capabilities are provided:

- o BSC multi-leaving protocol
- o EBCDIC transparency
- o Data compression/expansion
- o Switched or dedicated terminal communication facilities

HOST RESIDENT FACILITY

Honeywell's Level 66 Host Resident Facility (HRF) is a set of programs that run on a Level 66 system under the control of GCOS. The HRF permits Level 6 users to write, compile, assemble, and link Level 6 programs on a Level 66. Once developed, HRF generated programs can be sent to a Level 6 using the Level 6/Level 66 File Transmission Facility and executed at the convenience of the Level 6 operator. HRF includes a facility to permit the printing of Level 6 memory dumps on a Level 66 line printer.

APPENDIX C
HARDWARE SUPPORTED

HARDWARE RESOURCES

Figure C-1 shows the hardware resources that can be used in a Level 6 configuration. (Minimum configurations are given below.) For a complete description of central processors, peripheral and communications hardware, refer to the Level 6 Minicomputer Handbook.

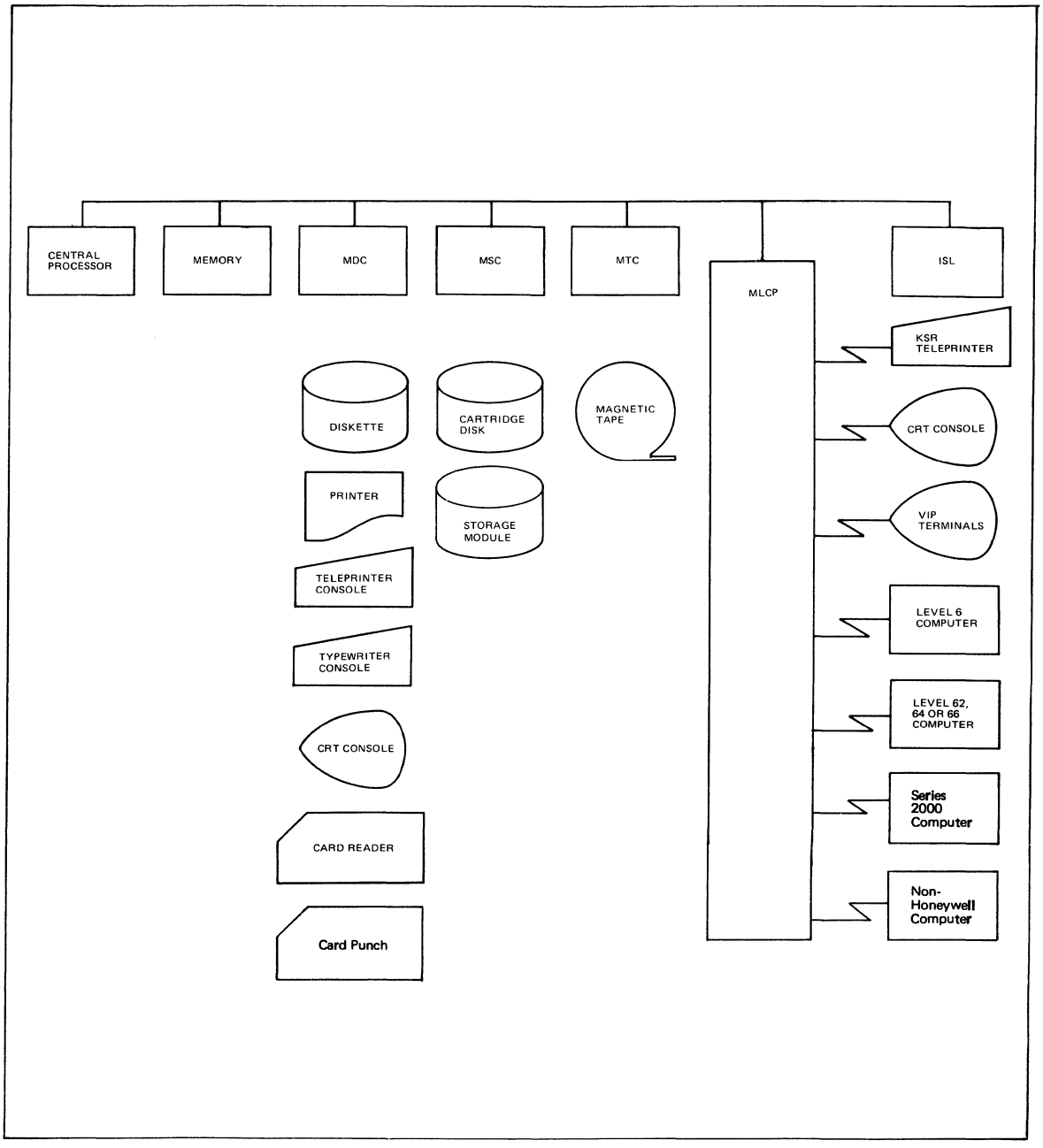

Figure C-1. Level 6 Hardware

Memory is available in multiples of 8K words up to 1024K words, depending on the central processor. A multiple device controller (MDC) controls terminals, printers, diskettes, and card readers. A mass storage controller (MSC) controls fixed and removable cartridge disks, and storage modules. A magnetic tape controller (MTC) controls 7-and 9-track magnetic tape devices. The terminals that can be connected to the MDC are: keyboard-send-receive (KSR) and automatic-send-receive (ASR) teleprinters (no paper tape); cathode ray tube keyboard console (CRT); and typewriter console.

The multiline communications processor (MLCP) is a programmable communications controller that is programmed to handle supported asynchronous terminals, synchronous terminals, and communications to other computers. The MLCP can be programmed by the user to handle devices not supported by Honeywell-supplied software.

The asynchronous terminals supported on the MLCP are: KSR teleprinters, CRT consoles, and visual information projection (VIP) system terminals with line editing at the keyboard, buffered line transmission, and full cursor control to edit text from the keyboard.

The synchronous terminals supported on the MLCP are: VIP terminals with keyboard, screen, and receive-only printer (ROP) that can be used in a poll/select mode of operation. The VIP is an interactive display terminal that, in addition to line cursor control, provides forms control to display a form on the screen for formatted data entry, and function code keys to transmit a function code to be interpreted by the receiver.

The synchronous BSC2780 communications protocol is used for communicating with other Level 6 computers, Level 66 computers, and non-Honeywell computers.

Figure C-1 also illustrates that data transfers occur over communications lines that are either hard-wired, dedicated, or dial up.

EQUIPMENT REQUIREMENTS

Minimum Equipment for Program Preparation

The following equipment is required for program preparation:

- One Level 6 central processor with full or basic control panel and 32K words of memory (SAF mode) or 40K words of memory (LAF mode).
- Operator terminal (with appropriate device-pac) connected through the multiple device controller or the multiline communications processor.[1]
- One million bytes disk storage. Diskettes, cartridge disk, or storage module can be used. Diskettes must be connected through the multiple device controller and appropriate device-pac. Cartridge disks must be connected through the mass storage controller and appropriate device-pac. Storage modules must be connected through the storage module controller and adapter.[1]

Minimum Equipment for Online Applications

The following equipment is required to run online applications:

- One Level 6 central processor with full or basic control panel and the following memory: 24K words of memory (SAF mode); 32K words of memory (SAF mode with communications); 32K words of memory (LAF mode); 40K words of memory (LAF mode with communications).
- Operator terminal (with appropriate device-pac) connected through the multiple device controller or the multiline communications processor.[1]
- One-quarter million bytes disk storage. Diskettes, cartridge disk, or storage module can be used. Diskettes must be connected through the multiple device controller and appropriate device-pac. Cartridge disks must be connected through the mass storage controller and appropriate device-pac. Storage Modules must be connected through the storage module controller and adapter.[1]

[1] A Multiple Device Controller is not available on Model 23 systems; the peripheral interface is achieved through adapters only. A Multiline Communications Processor is not available on Model 23 systems; the communications interface is achieved through a Dual-Line Communications Processor.

Hardware Supported

Table C-1 lists Level 6 equipment and options supported by the GCOS6 Mod 400 Operating System.[2]

TABLE C-1. HARDWARE SUPPORTED

Category	Marketing Identifier	Description
Central Processors	CPS9350 CPS9351 CPS9352 CPS9353 CPS9354 CPS9355 CPS9356 CPS9357 CPS9358 CPS9359	Model 23 Central Processor
	CPS9450 CPS9451 CPS9452 CPS9453	Model 6/34 Central Processors
	CPS9470 CPS9471 CPS9472 CPS9473	Model 33 Central Processors
	CPS9560 CPS9561 CPS9562 CPS9563	Model 43 Central Processors
	CPS9566 CPS9567	Model 47 Central Processors
	CPS9570	Model 53 Central Processors
	CPS9572	Model 57 Central Processor
Memory		The user can add core memory, MOS single-fetch, MOS double-fetch, or high density memory as desired, up to the limits of the central processor.
Central Processor Options	CPF9503 CPF9504	Scientific Instruction Processor Portable Plug-in Control Panel The memory management unit and the watchdog timer are both supported. These items, depending upon processor, are available as standard equipment, a factory-installed option, or as part of a field upgrade module.

[2]Table C-1 is not a complete listing of Level 6 equipment. In some cases (e.g., cabinetry, cables), equipment is not listed because its use is invisible to the software; in other cases, equipment is not listed because full software qualification had not been performed at the time this document was published. Information relative to the operation of GCOS6 Mod 400 on equipment not listed can be obtained from Honeywell marketing representatives. The user should also note that Table C-1 contains no information with respect to installation prerequisites; Table C-1 cannot be used to configure end-user systems.

TABLE C-1 (CONT). HARDWARE SUPPORTED

Category	Marketing Identifier	Description
Multiple Device Controller (not required on Model 23)	MDC9101	Multiple device controller
Console Device-Pac (Adapter)	KCM9101	Keyboard console device-pac
	KCM9301	Single Console Adapter (Model 23 only)
	KCM9302	Dual Console Adapter (Model 23 only)
Console Devices	TTU9101	Teleprinter console (ASR33)
	TTU9102	Teleprinter console (KSR33)
	DKU9101	CRT/keyboard console, 64-character set
	DKU9102	CRT/keyboard console, 96-character set
	DKU9103	CRT/keyboard console, 64-character set
	DKU9104	CRT/keyboard console (VIP7205), 96-character set
	TWU9101	Typewriter console, 30 characters-per-second, 64-character set
	TWU9104	Typewriter console, 30 characters-per-second, 96-character set
	TWU9106	Typewriter console, 120 characters-per-second, 96-character set
	VIP7200	CRT/keyboard console
	VIP7100	CRT/keyboard console
Printer Device-Pac (Adapter)	PRM9101	Printer device-pac
	PRM9301	Printer Adapter (Model 23 only)
Printers	*Line Printers*	
	PRU9103	240 lpm, 96-character set
	PRU9104	300 lpm, 64-character set
	PRU9105	480 lpm, 96-character set
	PRU9106	600 lpm, 64-character set
	PRU9108	660 lpm, 96-character set
	PRU9109	900 lpm, 64-character set
	PRF9102	12-channel vertical format unit option for PRUs 9103, 9104, 9105, 9106, 9109
	Serial Printers	
	PRU9101	60 lpm, 64-character set
	PRU9102	60 lpm, 96-character set
	PRU9112	120 cps, 96-character set (Lina 21)
	PRU9114	160 cps, 96-character set
Card Reader Device-Pac (Adapter)	CRM9101	Card reader device-pac

TABLE C-1 (CONT). HARDWARE SUPPORTED

Category	Marketing Identifier	Description
Card Readers	CRU9101	300 cpm, punched card
	CRU9102	300 cpm, punched and marked sense card
	CRU9103	500 cpm, punched card
	CRU9104	500 cpm, punched and marked sense card
	CRF9101	51-column option for CRUs 9101, 9102, 9103, 9104
	CRU9108	300 cpm, punched card reader
	CRU9109	300 cpm, punched and IBM marked sense card reader
	CRU9110	300 cpm, punched and HIS marked sense card reader
	CRU9111	500 cpm, punched card reader
	CRU9112	500 cpm, punched and IBM marked sense card reader
	CRU9113	500 cpm, punched and HIS marked sense card reader
Card Punch and Card Reader/ Punch Device-Pac (Adapter)	CRM9103	Device-pac for card punch and card reader punch
Card Punch	PCU9101	100 cpm card punch
Card Reader/ Punch	CCU9101	400/100 card reader/punch
Diskette Device-Pac (Adapter)	DIM9301	Diskette Adapter (Model 23 only)
	DIM9101	Diskette device-pac adapter for single-sided diskettes
	DIM9102	Diskette device-pac adapter for double-sided diskettes
Diskettes	DIU9101	Single diskette (single-sided), 256K-Byte
	DIU9102	Dual diskette (single-sided), 512K-Byte
	DIU9103	Single diskette (double-sided), 512K-Byte
	DIU9104	Dual diskette (double-sided), 1MK-Byte
Cartridge Disk Controller	MSC9101	Cartridge disk controller
Cartridge Disk Device-Pac (Adapter)	CDM9101	Cartridge disk device-pac
Cartridge Disk	CDU9101	Low density 100 tpi removable disk (1.25 million words)
	CDU9102	Low density 100 tpi fixed and removable disks (2.5 million words)
	CDU9103	High density 200 tpi removable disk (2.5 million words) CDU9104
	CDU9104	High density 200 tpi fixed and removable disks (5.0 million words)
	CDU9114	100 tpi, low density, fixed and removable disks (2.5 million words)
	CDU9116	200 tpi, high density, fixed and removable disks (5.0 million words)
	CDU9115	High density removable disk (2.5 million words)
Mass Storage (Storage Module) Controller and Device-Pac	MSC9102	Mass Storage Controller/Device Pac This equipment used for connection of storage modules; device-pac is included as part of controller.

TABLE C-1 (CONT). HARDWARE SUPPORTED

Category	Marketing Identifier	Description
Mass Storage Devices (Storage Modules)	MSU9101	40 megabyte, 411 cylinders
	MSU9105	40 megabyte, 411 cylinders
	MSU9102	80 megabyte, 823 cylinders
	MSU9103	143/127 megabyte
	MSU9104	288/256 megabyte
	MSU9106	80 megabyte, 823 cylinders
Magnetic Tape Controllers	MTC9101	Magnetic tape controller for NRZI drives
	MTC9102	Magnetic tape controller for PE/NRZI drives
Magnetic Tape Device-Pacs	MTM9101	Device-pac for 7-track drives
	MTM9102	Device-pac for 9-track drives
Magnetic Tape Drives	MTU9104	Magnetic tape drive (9-track NRZI, 45 ips)
	MTU9105	Magnetic tape drive (9-track NRZI, 75 ips)
	MTU9109	Magnetic tape drive (9-track NRZI/PE, 45 ips)
	MTU9110	Magnetic tape drive (9-track NRZI/PE, 75 ips)
	MTU9114	Magnetic tape drive (9-track PE only, 45 ips)
	MTU9115	Magnetic tape drive (9-track PE only, 75 ips)
	MTU9116	Magnetic tape drive (9-track NRZI, 45 ips)
	MTU9117	Magnetic tape drive (9-track NRZI, 75 ips)
	MTU9112	Magnetic tape drive (7-track NRZI, 45 ips)
	MTU9113	Magnetic tape drive (7-track NRZI, 75 ips)
	MTU9120	Magnetic tape drive (7-track NRZI, 45 ips)
	MTU9121	Magnetic tape drive (7-track NRZI, 75 ips)
Multiline Communications Processor	MLC9101	With Communications-Pac for eight asynchronous lines
	MLC9102	With Communications-Pac for eight synchronous lines
	MLC9103	Multiline Communications Processor only – required Communications-Pac(s) depending on choice of line speeds.
Communications-Pacs (Adapters)	DCM9101	Communications-Pac, two asynchronous lines, with cable
	DCM9102	Communications-Pac, one asynchronous line, with cable
	DCM9103	Communications-Pac, two synchronous lines, with cable
	DCM9104	Communications-Pac, one synchronous line, with cable
	DCM9110	Communications-Pac – Autocall unit for one or two synchronous or asynchronous lines
	DCM9106	Communications-Pac, one synchronous HDLC line, with cable (for RBF)
	DCM9112	Communications-Pac, broadband HDLC line (Bell 301, 303 compatible to 72 KB)
	DCM9113	Communications-Pac, broadband HDLC line (CCITT/V35 compatible to 72 KB)
	DCM9115	Communications-Pac, broadband synchronous line, (MIL 188C compatible to 72 KB)
	DCM9116	Communications-Pac, dual asynchronous line, (MIL 188C compatible to 96 KB)
	DCM9120	Communications-Pac, HDLC MIL 188C

TABLE C-1 (CONT). HARDWARE SUPPORTED

Category	Marketing Identifier	Description
Dual-Line Communications Adapter (Model 23 only)	DCM9301	Controls up to two asynchronous lines
	DCM9302	Controls one asynchronous line
	DCM9303	Controls up to two synchronous lines
	DCM9304	Controls one synchronous line
Terminals	*Asynchronous Terminals*	
	ASR-33	Keyboard/printer, 110 baud
	ASR-35	Keyboard/printer, 110 baud
	KSR-33	Keyboard/printer, 110 baud
	VIP7100	CRT/keyboard, up to 9600 baud
	VIP7200	CRT/keyboard, with cursor control, line editing and buffered transmission, up to 9600 baud
	VIP7205	CRT/keyboard/display terminal, 96-character set
	TWU1001	Keyboard/printer 30 cps
	TWU1003	Keyboard/printer 30 cps
	TWU1005	Keyboard/printer 120 cps
	PRU1001	Printer Terminal 30 cps
	PRU1003	Printer Terminal 30 cps
	PRU1005	Printer Terminal 120 cps
	VIP 7800	CRT/Keyboard/display terminal
	Synchronous Terminals	
	VIP7700R	Nonpolled CRT/keyboard with optional ROP; 2000 to 4800 baud
	VIP7700R	Polled CRT/keyboard with optional ROP; 2000 to 4800 baud
	VIP7760-2A	Polled CRT/keyboard with optional ROP; 2000 to 4800 baud
	VIP7700	Nonpolled CRT/keyboard with optional ROP; 2000 to 4800 baud
	VIP7700	Polled CRT/keyboard with optional ROP; 2000 to 4800 baud
	Receive-only printer for CRT terminals	
	TN300	Printer Terminal 30 cps
	TN1200	Printer Terminal 120 cps
	PRU1001	Printer Terminal 30 cps (available on VIP7100 and VIP7200 only)
	PRU1003	Printer Terminal 30 cps
	PRU1005	Printer Terminal 120 cps
Modems	Modems supported for asynchronous communications terminals are: o 103,[1] 113,[1] 202 type modems (these modems must be equipped with the option to disconnect the data set after a carrier drop of 110 milliseconds) For synchronous communications terminals they are: o 201, 203, 208 type modems Also supported are: o Honeywell modem by-pass for both asynchronous and synchronous terminals o Modem types where the connection, disconnection and dataset control settings can be user specified A modem is not required for a direct connect asynchronous terminal or synchronous terminal with a timing source in the terminal or in the MLCP.	

System Builder Products

Intersystem Link (GIS9010) and Writable Control Store (CPF9509) are offered for system builder use only. Although Intersystem Link and Writable Control Store are usable under GCOS6 Mod 400, full end-user software support is not available.

byte
A sequence of eight consecutive binary digits operated upon as a unit.

calling sequence
A standard code sequence by which system services or external procedures are invoked.

CCP
See channel control program.

channel control program (CCP)
A program that resides in the MLCP and processes characters, protocol headers, and framing characters.

CI
See control interval.

clock frequency
The line frequency, in cycles per second, that is the basis (coupled with the scan cycle) for calculating the interval between real time clock-generated interrupts.

Clock Manager
A Monitor component that handles all requests to control tasks based on real-time considerations, and requests for the time-of-day and date in ASCII format.

clock request block
A control structure supplied by a task to request a service from the Clock Manager.

clock scan cycle
The time in milliseconds between clock-generated interrupts.

clock timer block
The control structure used by the Clock Manager to control the clock-related processing of tasks.

command
An order that is processed by the command processor.

command input file (command-in)
Any file or device from which commands to the command processor are read.

command language
The set of commands that can be issued by a user to control the execution of the user's online or batch task.

command level
The state of the command processor, when it is capable of accepting commands, indicated by the display of the RDY (ready) message.

command processor
A software component that interprets control commands issued by the operator or a user, and invokes the required function.

commercial simulator
A software component that executes a set of business-oriented instructions.

communications device
A device that transfers data over communications lines and is connected through the MLCP.

concurrency

The read or write file access that the reserving task group intends for its tasks and the read or write file access that the reserving task group allows to other task groups.

configuration

The procedure that involves the use of configuration directives to define a system that corresponds to actual installation hardware.

control interval (CI)

A logical unit of transfer between main memory and a disk device; the size is specified by the user and remains constant for a file. The CI determines the buffer size.

CTB

See clock times block.

CRB

See clock request block.

device driver

A software component that controls all data transfers to or from a peripheral or communications device.

device-pac

The adapter between an MSC or MDC controller and peripheral device (e.g., printer, diskette drive).

direct access

The method for reading or writing a record in a file by supplying its key value.

directive

A "secondary" level order read through the user-in file to a "secondary" processor. Examples are Editor, Linker, Patch, Debug, and CLM (configuration) directives.

directory

A structure in a volume directory containing a description of a file or another directory.

disk

A generic name for mass storage devices such as diskette, cartridge disk, and storage module.

dormant state

A task is in the dormant state when there is no current request for the task.

entry point

A symbolic start address within the root segment of a bound unit.

equal name convention

A special pathname convention that can be used with certain commands to automatically construct the output pathname entry name when the input pathname entry name has been resolved.

error output file (error-out)

The file or device by which the system communicates error information to the user or operator; established when a group request is entered.

exclusive online pool

A memory pool whose boundaries do not overlap those of other pools.

expandable online pool

An online pool that may expand into the batch pool space.

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 6) GCOS 6 MOD 400 SYSTEM CONCEPTS ADDENDUM A

ORDER NO. CB20A, REV.0

DATED JUNE 1978

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

Honeywell

CUT ALONG LINE

FOLD ALONG LINE

FOLD ALONG LINE