HONEYWELL

DPS 6
GCOS 6 MOD 400
SYSTEM CONCEPTS

SOFTWARE

DPS 6
GCOS 6 MOD 400
SYSTEM CONCEPTS

SUBJECT

System Concepts for GCOS 6 MOD 400

SPECIAL INSTRUCTIONS

This manual supersedes *DPS 6 GCOS 6 MOD 400 System Concepts* (Order No. CZ03-00), dated December 1982. Sections 4 and 5 have been restructured. The information that formerly resided in Section 6 has been made part of Section 5. Former Section 7 has been renumbered as Section 6. Wherever possible, change bars are used to indicate new and changed information, and asterisks are used to denote deletions.

SOFTWARE SUPPORTED

This manual supports Release 4.0 of the MOD 400 Executive.

ORDER NUMBER

CZ03-01

March 1986

Honeywell

PREFACE

This manual is written for all users of the MOD 400 operating system.

It will prove particularly informative to those responsible for building MOD 400 systems and those who design application programs and/or system functionality other than that supplied by Honeywell.

This manual contains a general description of the way in which processing is performed on MOD 400 systems. It presents a discussion of the MOD 400 Executive in terms of its design concepts and processing functionality. Not discussed are such topics as equipment lists, available software, and supporting manuals. No detailed procedural information is discussed; several procedures are, however, outlined.

The major topics discussed are:

- File system, including file and pathname concepts, file protection, and buffering operations.
- System access path including login, user registration, and the command environment.
- Execution environment, including a description of tasks, task groups, memory usage, and bound units.
- Task execution, including priority levels, logical resource numbers, and deferred processsing facilities.

USER COMMENTS FORMS are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

File No.: 1R13, 1S13

CZ03-01

- Backup and recovery facilities, including the backup and restoration of disk files, the preservation of the execution environment during a power failure, the recovery of files at the record level, and the recovery and restart of task groups.

Although no manual is prerequisite to this manual, you may find it convenient to have read the Software and Documentation Directory. *

Each section/appendix of this document is structured according to the heading hierarchy shown below. Each heading indicates the relative level of the text that follows it.

Level	Heading Format
1 (highest)	ALL CAPITAL LETTERS, UNDERLINED
2	Initial Capital Letters, underlined
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, not underlined

MANUAL DIRECTORY

The following publications constitute the GCOS 6 MOD 400 manual set. See the "Software/Manual Matrix" of the Guide to Software Documentation for the current revision number and addenda (if any) of the manuals.

Manuals are obtained by submitting a Honeywell Publications Order Form to the following address:

Honeywell Information Systems Inc.
47 Harvard Street
Westwood, MA 02090
Attn: Publications Services

Honeywell software reference manuals are periodically updated to support enhancements and improvements to the software. Before ordering any manual listed below, the customer should refer to the Guide to Software Documentation to obtain information concerning the specific edition of the manual that supports the software currently in use at the installation. When specifying manuals on the Publications Order Form, a customer using the 4-digit base publication number listed below will obtain the latest edition of the manual currently in stock. The Publications Distribution Center can provide specific editions of a publication only when supplied with the 7- or 8-character order number described in the Guide to Software Documentation.

Honeywell applications software packages - such as INFO, the Honeywell Manufacturing System (HMS), and TPS 6 - provide specialized services. See your Honeywell representative for information concerning the availability of applications software and supporting documentation.

Base Publication Number	Manual Title
CW35	GCOS 6 C User's Guide
CZ01	GCOS 6 MOD 400 Guide to Software Documentation
CZ02	GCOS 6 MOD 400 System Building and Administration
CZ03	GCOS 6 MOD 400 System Concepts
CZ04	GCOS 6 MOD 400 System User's Guide
CZ05	GCOS 6 MOD 400 System Programmer's Guide - Volume I
CZ06	GCOS 6 MOD 400 System Programmer's Guide - Volume II
CZ07	GCOS 6 MOD 400 Programmer's Pocket Guide
CZ09	GCOS 6 MOD 400 System Maintenance Facility Administrator's Guide
CZ10	GCOS 6 MOD 400 Menu System User's Guide
CZ11	GCOS 6 MOD 400 Software Installation Guide
CZ15	GCOS 6 MOD 400 Application Developer's Guide
CZ16	GCOS 6 MOD 400 System Messages
CZ17	GCOS 6 MOD 400 Commands
CZ18	GCOS 6 Sort/Merge
CZ19	GCOS 6 Data File Organizations and Formats
CZ20	GCOS 6 MOD 400 Transaction Control Language Facility
CZ21	GCOS 6 MOD 400 Display Formatting and Control
CZ22	GCOS 6 VISION Reference Manual
CZ23	DM6 AZ7 Reference Card
CZ24	Introduction to DM6 AZ7 Query Writing
CZ25	DM6 AZ7 Reference Manual
CZ29	GCOS 6 VISION Reference Card
CZ31	GCOS 6 Advanced COBOL Compiler User's Guide
CZ32	GCOS 6 Multiuser COBOL Compiler Guide
CZ34	GCOS 6 COBOL 74 Language Reference
CZ35	GCOS 6 COBOL Quick Reference Guide
CZ36	GCOS 6 BASIC Reference
CZ37	GCOS 6 BASIC Quick Reference Guide
CZ38	GCOS 6 Assembly Language (MAP) Reference
CZ39	GCOS 6 Advanced FORTRAN Reference
CZ40	GCOS 6 Pascal User's Guide
CZ42	GCOS 6 Ada Compiler System User's Guide
CZ52	DM6 I-D-S/II Programmer's Guide
CZ53	DM6 I-D-S/II Data Base Administrator's Guide
CZ54	DM6 I-D-S/II Reference Card
CZ70	Electronic Mail Facility Administrator's Guide
CZ71	DM6 TP Development Reference
CZ72	DM6 TP Application User's Guide
CZ73	DM6 TP Forms Processing
CZ74	GCOS 6 Data Base Augmented Real-Time Tracing System User's Guide
CZ93	Electronic Mail Facility User's Guide
GZ13	GCOS 6 MOD 400 Release 4.0 Migration Guide

Base Publication Number	Manual Title
HC01	MOD 400 Application Development Overview
HC12	Disk-Based Data Entry Facility-II User's Guide
HC13	Disk-Based Data Entry Facility-II Operator's Quick Reference Guide

The following manuals describe the MOD 400 distributed processing software components:

Base Publication Number	Manual Title
CB35	DPS 6/DPS 7 PVE File Transfer Facility User's Guide
CF11	DPS 6/DPS 7 PVE Remote Batch Facility User's Guide
CG90	Interactive Entry Facility-II User's Guide
CZ59	Level 6 to Level 6 File Transmission Facility User's Guide
CZ60	Level 6 to Level 66 File Transmission Facility User's Guide
CZ61	Level 6 to Level 62 File Transmission Facility User's Guide
CZ62	BSC Transport Facility User's Guide
CZ63	2780/3780 Workstation Facility User's Guide
CZ64	HASP Workstation Facility User's Guide
CZ65	Programmable Facility/3271 User's Guide
CZ66	Remote Batch Facility/66 User's Guide
GG19	Disk-Based VIP7305 Emulator Facility User's Guide
GG20	Asynchronous Communications Facility User's Guide
GT18	Disk-Based VIP7705 Emulator Facility User's Guide
GT19	Disk-Based VIP7814 Emulator Facility User's Guide

The following manuals describe the ORACLE data base management facility:

Base Publication Number	Manual Title
GS61	GCOS 6 MOD 400 ORACLE Installation Guide
GS62	GCOS 6 MOD 400 ORACLE Data Base Administrator's Guide
GS63	GCOS 6 MOD 400 ORACLE Interactive Application Facility (IAF) Terminal Operator's Guide
GS64	GCOS 6 MOD 400 ORACLE Interactive Application Facility (IAF) Terminal Operator's Reference Manual
GS65	GCOS 6 MOD 400 ORACLE Interactive Application Facility (IAF) Designer's Guide
GS66	GCOS 6 MOD 400 ORACLE Interactive Application Facility (IAF) Designer's Reference Manual
GS67	GCOS 6 MOD 400 ORACLE HLI Precompiler Interface
GS68	GCOS 6 MOD 400 ORACLE Host Language Call Interface Manual
GS69	GCOS 6 MOD 400 ORACLE RPF Report Text Formatter User's Guide
GS70	GCOS 6 MOD 400 ORACLE RPT Report Generator User's Guide
GS71	GCOS 6 MOD 400 ORACLE SQL/UFI Reference Manual
GS72	GCOS 6 MOD 400 ORACLE Terminal User's Guide
GS73	GCOS 6 MOD 400 ORACLE Utilities Manual
GS74	GCOS 6 MOD 400 ORACLE Error Messages and Codes

In addition, the following publications provide supplementary information:

Base Publication Number	Manual Title
AS22	Level 6 Models 6/34, 6/36, and 6/43 Minicomputer Handbook
AT97	Level 6 Communications Handbook
CC71	Level 6 Minicomputer Systems Handbook
CD18	Level 6 MOD 400/600 Online Test and Verification Operator's Guide
FQ41	Writeable Control Store User's Guide

These five manuals are not covered by the Guide to Software Documentation. See your Honeywell representative for information concerning the versions of the manuals relevant to your installation.

Users should be aware that a software release bulletin accompanies each software product ordered from Honeywell. Users should consult the software release bulletin before using the software. Users should contact their Honeywell representative if a copy of the software release bulletin is not available.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

ILLUSTRATIONS

TABLES

Section 1
SYSTEM CHARACTERISTICS

GCOS 6 MOD 400 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 and other non-interactive applications can be performed concurrently in multiple absentee streams.

MOD 400 is a multifunctional system capable of supporting a variety of processing functions. You can develop and execute applications software, perform forms data entry, transmit files to other DPS 6 computers, and enter jobs for execution at remote sites.

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

OPERATING FACILITIES

MOD 400 supports multiprogramming, the 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 it shares the resources of the system.

If you define the environment to run more than one application task group concurrently, you are multiprogramming. In this environment you can execute each task in a task group sequentially, or concurrently (which is multitasking). You can run multiple online and absentee task groups concurrently.

The number of task groups that can run is limited by the number of central processors in your system, by central processor power, and by the amount of memory available. Concurrently executing task groups can occupy independent dedicated memory areas, or they can 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. MOD 400 allocates memory dynamically from pools and can relocate programs at load time. Once a task group requests execution, its tasks are dispatched according to their assigned priority levels. In a multiprocessor system, a task is dispatched when a central processor becomes free. When more than one task shares a priority level, tasks are serviced in round-robin fashion.

Use of disk files 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 of the containing directory or file. File reference can be 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.

SOFTWARE FACILITIES

MOD 400 offers you a comprehensive set of software components that perform a wide variety of functions. The following paragraphs briefly describe these software components.

System Control Software

System control software includes:

- Task Manager: Handles the disposition of tasks within the system's central processor(s) and responds to requests placed against tasks. The Task Manager 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.

- Swapper: Controls the allocation of swap pool memory and swap file space. Swaps tasks out when swap pool memory is required and swaps them back when the memory is available.

- Memory Manager: Controls dynamic requests for memory and the return of memory to group work segments. Also controls the allocation of all memory in independent (non-swapped) pools and of task groups assigned to the swap 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.

- Operator Interface Manager: Manages all messages sent by task groups to the operator terminal or from the operator terminal to task groups.

- Loader: Loads the root and overlays of a bound unit into memory from a disk.

- Listener: Monitors a selected set of local and remote terminals. When you enter a Login command requesting access to the system at one of the terminals, the Listener causes a task group to be spawned for you.

- Command Processor: Processes all commands. The Command Processor must be the lead task of an absentee task group and can be the lead task of any other task group.

- User Productivity Facility (Menu Subsystem): Provides you with a screen-oriented interface to the Executive and to applications.

- Message Facility: Provides a means for sending and receiving messages between tasks and between task groups.

- Message Reporter: Extracts messages from the message library, formats them, and delivers them to a user-specified location such as a terminal, a program buffer, etc.

- Error Logging Facility: Provides a mechanism for accumulating statistics on memory errors and peripheral devices. Should the error-per-use ratio exceed a specified threshold, a warning message is sent to the operator terminal.

File System Software

MOD 400 provides software to handle Input/Output (I/O) functions of each of the supported devices. The File System software is designed to work in conjunction with the data management conventions established for each device. The File System software is available through system commands or, for an Assembly language program, through system service macrocalls.

Utility Software

The system provides a comprehensive set of utility programs for performing frequently used programming functions. The system programs used by MOD 400 for the various utility functions are invoked by system commands.

Program Development Software

MOD 400 supports a large set of program preparation components, utilities, and debugging aids for application development. Programming languages include PASCAL, FORTRAN, COBOL, C, Ada, BASIC and Assembly language. Display formatting and control facilities provide the means for developing, using, and maintaining terminal display forms.

Data Communications Software

MOD 400 supports four levels of communications interface. Terminals and/or remote host computers can be accessed through the:

- Sequential file interface of the File System software
- Display formatting and control software
- Physical I/O interface of the system
- Various distributed systems facilities.

Specialized software components called Line Protocol Handlers (LPHs) support the different device classes and the various conventions established for data transfer.

Distributed Systems Software

MOD 400 supports software packages that permit use of DPS 6 in a distributed processing environment. Using the packages provided by Honeywell, your DPS 6 system can become a node on a network and can communicate with DPS 6, DPS 7, DPS 8, and other systems across a variety of links.

The Distributed Systems Architecture 6 (DSA6) package follows the layered structure of the Open Systems Interconnection (OSI) defined by the International Standards Organization. DSA6 is a set of networking products that includes a transport facility, a network terminal manager, a unified file transfer facility, a remote file facility, a remote batch facility, and an application interface facility. DSA6 also supports terminal access to IBM-hosted applications through the DSA/SNA gateway.

The Systems Network Architecture 6 (SNA6) package emulates most operations of standard IBM devices so that DPS 6 systems can interface with an IBM SNA network. SNA6 provides a remote job entry facility, a file transmission facility, an interactive terminal facility, and an application interface facility.

Data Management Software

MOD 400 supports data base management, query and report writing, and transaction processing software packages. Data base management packages are available for relational and network data bases. Query and report writing packages allow you to retrieve information from all supported data bases. Transaction processing packages support standalone systems as well as applications connecting to remote host processors through the Distributed Systems software.

Data Entry Software

MOD 400 supports a multistation, forms-oriented source data collection capability. The Data Entry Facility-II (DEF-II) package embodies established data entry concepts in a menu-driven approach, making it easy to specialize and run procedures. Data collected and validated by DEF-II can be organized into a file and transferred to another system through the Distributed Systems software.

Office Automation Software

MOD 400 supports the Office Automation System (OAS) facility. OAS offers a wide range of office processing functions including document processing, electronic mail, document transfer, records processing, spreadsheets, communications, and file management.

Section 2
FILE CONCEPTS

A file is a logical unit of data composed of a collection of records. The principal external devices available for storing files are:

- Disk devices (for example, diskettes, cartridge disks, cartridge module disks, fixed (sealed) disks, and mass storage units)
- Magnetic tape units (for example, 1/2-inch tapes and 1/4-inch cartridge tapes).

These external devices are referred to as volumes (for example, disk volume, tape volume).

Various conventions have been established to identify and locate files stored on disk and magnetic tape. These conventions facilitate the orderly and efficient use of the data stored on the volumes.

Unit record devices (such as card readers, card punches, printers, and terminals) also use the file concepts. However, since unit record devices cannot be used to store files, there is less need to establish conventions for identification and location. A unit record file is simply the data that is read or written at any one time (for example, a line entered at a terminal).

DISK FILE CONVENTIONS

You must be able to specify an access path to any given file on a disk volume that contains multiple files. Files must therefore be organized on the volume in some predictable fashion. The MOD 400 File System provides a set of volume organization conventions by which the system can locate any element that resides on the volume.

The principle elements of this organization, aside from the files themselves, are directories. The access path to any given element on a volume is known as a pathname.

Directories

Files on disk devices are logically arranged by the File System in a tree-structured hierarchy. The basic elements of this hierarchy are special files known as directories. The directories are used to point to the location of data files, which are the endpoints of the tree structure.

A directory on a disk volume is an index that contains the names and starting locations (sector numbers on the volume) of files or other directories (or both). The elements in the directory are said to be "contained in" or "subordinate to" the directory. Therefore, the organization of a disk volume is a multilevel structure. The complexity of the access path to any given element in the structure depends on the number of directories between the root and the desired element.

A sample directory structure is illustrated in Figure 2-1. The base directory on a volume is termed a root directory. In Figure 2-1 the root directory is VOL01. Root directory VOL01 contains two subordinate directories, DIR1 and DIR2. Directories DIR1 and DIR2, in turn, contain data files FILEA, FILEB, FILEC, and FILED.

The root directory and other special types of directories are described in the following paragraphs.

ROOT DIRECTORY

The File System maintains a tree structure for each disk mounted at any given time. At the base of each tree structure is a directory known as the root directory. This is the directory that ultimately contains every element that resides on the volume, either immediately or indirectly subordinate to it. The root directory name is the same as the volume identifier of the volume on which it resides. The directory VOL01 in Figure 2-1 is the root directory on the disk volume VOL01.

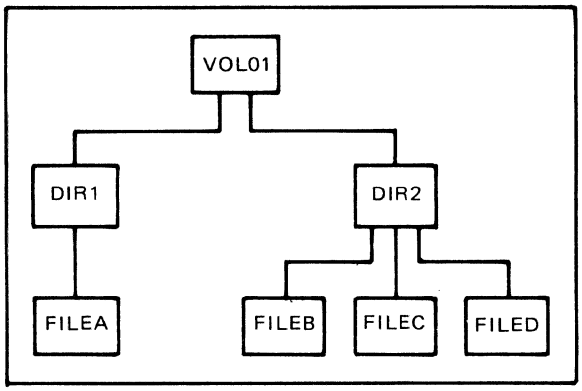


Figure 2-1. Example of Disk Directory Structure

SYSTEM ROOT DIRECTORY

One or more disk root directories can be known to the system at any time during its operation. One of these, the System Root Directory (SRD), is required at all times. The access paths of files in the SRD start with two greater-than signs (>>). The volume used by the operator to initialize the system establishes the SRD. The boot volume must contain the SRD; it also normally contains system programs, commands, and other routinely used elements. The SRD must contain a number of directories and files that the system needs to perform its functions, including >>Z3EXECUTIVEL, >>SID, >>AID, >>HIS, and >>USER_REG. For more information, refer to the System Building and Administration manual.

USER ROOT DIRECTORY

The File System can recognize one User Root Directory (URD), which you define through the Change System Directories command with the -ROOT argument. Files in the URD have access paths that start with a single greater-than character. The URD contains items such as UDD, LDD, MDD, FORMS, PROGS, and TRANS. For more information, refer to the System Building and Administration manual.

The URD and SRD can reside on different volumes or on the same volume. The installation can also have user volumes created to meet the installation's own particular needs. These volumes may contain user application programs and their associated data files, application program source and object code files, listing files, and anything else a user might want to store temporarily or permanently.

Refer to "Links" later in this section for information on another way to distribute software (system or user) onto more than one volume.

INTERMEDIATE DIRECTORIES

When you first create (format and name) a volume under the File System, it contains only a root directory. Within this directory, you can create any additional directories required to satisfy the needs of the installation. Consider, for example, a volume that is to contain data used by two application projects, each of which has several users associated with it. Each user has one or more files of interest to him or her. The volume has been initialized and contains a root directory name. Two directories can be created subordinate to the root directory, each identified by the project name. Then, subordinate to these directories, a directory can be created for each user associated with each project.

The data files are all contained within the personal directories. This sample directory structure is illustrated in Figure 2-2.

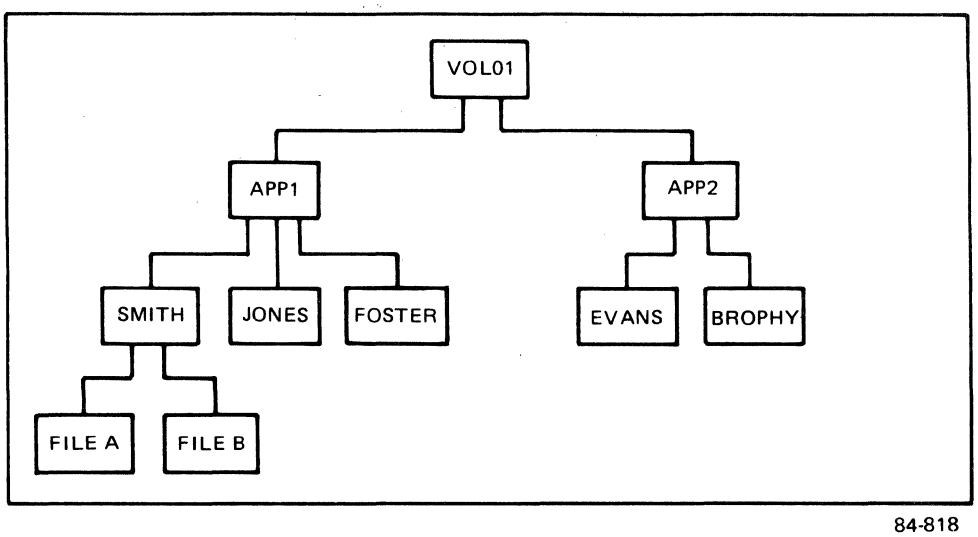


Figure 2-2. Sample Directory Structure

When the need for a user-created directory no longer exists, the directory can be deleted from the File System (deleted from the disk). The space it occupies, as well as the space occupied by its attributes in the immediately superior directory, is then available for reuse. A directory must be empty before it can be deleted. All directories and files subordinate to the one to be deleted must have been previously deleted by explicit commands.

WORKING DIRECTORY

The File System always starts at a root directory when it searches for a disk file or a directory. At times the search for an element residing on a disk volume may traverse a number of intermediate directory levels before the desired element is located, and the File System must be supplied with the names of all the directories it must pass on the way. Frequently all files of interest to a user doing work on the system are contained in a single directory that is three or four levels deep in the hierarchy. It is convenient to be able to refer to files in relation to a directory at some arbitrary level in the hierarchy rather than in relation to the root directory. The File System allows this to be done by recognizing a special kind of directory known as a working directory.

A working directory establishes a reference point that enables you to specify the name of a file or another directory in terms of its position relative to the working directory. If the access path of the working directory is made known to the File System, and if the desired element is contained in that directory, the element can be specified by just its name. The File System concatenates this name with the names of the elements of the working directory's access path to form the complete access path to the element.

Disk Directory and File Locations

The File System has total control over the physical location of space allocated to directories and files. You need never be concerned about where a directory or file resides on a volume. When a volume is first initialized, space is allocated to elements in essentially the order in which they are created. But, after the volume has been in use for some time, elements may have been deleted and the space they occupied made reusable. Then, when a new element is created, it is allocated the first available space. If more space is needed, it is obtained from the next free area.

Disk Directory and File Naming Conventions

Each disk directory and file name in the File System can consist of the following American Standard Code for Information Interchange (ASCII) characters:

- Uppercase and lowercase primary character set alphabetics (A-Z, a-z)
- Digits (0-9)
- Underscore (_)
- Hyphen (-)

- Period (.)
- Apostrophe (')
- Uppercase and lowercase characters whose hexadecimal equivalents are from C0-FE (Western European Latin alphabet, also called the extended character set).

The characters in the extended character set cannot be used in volume identifiers.

NOTE

If the terminal is not capable of processing 8-bit data, characters from the extended character set are displayed as periods or as their 7-bit equivalents.

When volumes, files, and directories are created, their identifiers are stored on disk exactly as entered, in uppercase and lowercase characters. For both the primary and extended character sets, MOD 400 considers uppercase and lowercase characters to be equivalent (for example, "DATA", "Data", and "data" all refer to the same file).

The first character of any name must not be character FF. The underscore character can be used to join two or more words that are to be interpreted as a single name (for example, DATE_TIME). The period character and one or more following alphabetic or numeric characters are normally interpreted as a suffix to a file name. This convention is followed, for example, by a compiler when it generates a file that is to be listed. The compiler identifies this file by creating a name of the form FILEA.L.

MAXIMUM NAME LENGTH

The name of a root directory (the volume identifier) can be from one through six characters in length. The names of other directories and files can be from 1 through 12 characters in length. The length of a file name must be such that any system-supplied suffix does not result in a name containing more than 12 characters.

UNIQUENESS OF NAMES

Within the system at any given time, the access path to every element must be unique. This requirement leads to the following rules for naming files:

- Only one volume with a given volume identifier can be mounted at any given time. (The system notifies you of an attempt to mount a volume having the same name as one already mounted.)

- Within a given directory, every immediately subordinate directory or file name must be unique. (The Create Directory and Create File commands notify you of an attempt to add a duplicate name.)

Note that uppercase/lowercase differences do not constitute uniqueness. As previously mentioned, "DATA", "Data", and "data" all refer to the same file.

Pathnames

The access path to any File System entity (directory or file) begins with a root directory name and proceeds through zero or more subdirectory levels to the desired entity. The series of directory names (and a file name if a file is the target entity) is known as the entity's pathname. The construction of a pathname is described below.

The total length of any pathname, including all symbols, cannot exceed 57 characters. A working directory pathname, however, cannot exceed 44 characters.

The last (or only) element in a pathname is the name of the entity upon which action is to be taken. This element can be a device name, directory name, or file name, depending on the function to be performed. For example, in the Create Directory command a pathname specifies the name of a directory to be created. The last element of this pathname is interpreted by the command as a directory name; any names preceding the final name are names of superior directories leading to it. An analogous situation occurs in the Create File command, except that in this case the final pathname element is the name of a file to be created.

SYMBOLS USED IN PATHNAMES

The following symbols are used to construct pathnames:

- Circumflex (^). Used at the beginning of a pathname to identify the name of a disk volume root directory (for example, ^VOL011).

- Circumflex Preceding Greater-Than Sign (^>). Used at the beginning of a pathname to identify the root directory of the current working directory (for example, ^>DIR1>FILEA is equivalent to ^VOL011>DIR1>FILEA if the current working directory is on VOL011).

- Greater-Than Sign (>). Used at the beginning of a pathname and between the names in a pathname.

- When used at the beginning of a pathname, the element whose name follows the > symbol is immediately subordinate to the root directory of the user root volume (it resides under the URD). Honeywell-supplied programs assume the URD contains the UDD, LDD, FORMS, MDD, PROGS, and TRANS directories.

  The correct way to refer to a directory in the URD is to precede the directory name by one greater-than sign (for example >UDD).

- When used between names in a pathname the > symbol indicates movement in the hierarchy away from the root directory. The symbol is used to connect two directory names or a directory name and a file name. Each occurrence of the > symbol denotes a change of one hierarchical level. The element to the right of the symbol is immediately subordinate to the element on the left.

  Reading a pathname from left to right thus indicates movement through the tree structure in a direction away from the root directory. For example, if the root ^VOL011 contains a directory named DIR1, the pathname of DIR1 is ^VOL011>DIR1. If the directory named DIR1 in turn contains a file named FILEA, the pathname of FILEA is ^VOL011>DIR1>FILEA

- Two Consecutive Greater-Than Signs (>>). Used at the beginning of a pathname to specify entities that are subordinate to the SRD. Honeywell-supplied programs assume the SRD contains the Z3EXECUTIVEL, SID, AID, HIS, and USER_REG directories.

  The correct way to refer to a directory in the SRD is to precede the directory name by two greater-than signs (for example >>SID).

  SYSLIB1 and SYSLIB2 can reside in either the SRD or the URD.

- Less-Than Sign (<). Used at the beginning of a pathname to indicate movement from the working directory toward the root directory. Consecutive symbols can be used to indicate changes of more than one level; each occurrence represents one level change. One or more less-than symbols may precede only a pathname that assumes a directory without actually referring to it explicitly. Such a pathname is called a relative pathname.

- ASCII Space Character (Hexadecimal 20). Used to indicate the end of an encoded pathname in a program. When represented in memory, a pathname must end with a space character.

The use of these symbols at the beginning of a pathname can be summarized as follows:

Symbol	Meaning
^volume	Any volume or root directory
^>	Under root of current working directory volume
>	Under URD root
>>	Under SRD root
<	Movement away from current working directory toward volume root

ABSOLUTE AND RELATIVE PATHNAMES

A full pathname is one that begins with a circumflex. A full pathname contains all necessary elements to describe a unique access path to a File System entity, regardless of the type and location of the device on which it resides or where your working or assumed directory is. You use a full pathname to locate directories and files that reside on a device other than that on which the system volume (the volume from which the system was initialized) is mounted.

The File System uses a full pathname when referring to a directory or file. However, it is frequently unnecessary for you to specify all of these elements. The File System can supply some of them when the missing elements are known to it and the abbreviated pathname is used in the appropriate context. An understanding of these conditions and contexts requires an understanding of absolute and relative pathnames.

Absolute Pathname

An absolute pathname is one that begins with a circumflex (^) or one or more greater-than symbols (>).

If an absolute pathname begins with a circumflex, it is a full pathname.

If an absolute pathname begins with one greater-than symbol, the first element named in the pathname is assumed to be immediately subordinate to the URD.

If the pathname begins with two greater-than symbols, the first element named in the pathname is assumed to be directly subordinate to the SRD.

Relative Pathname

A relative pathname is a shortened version of the absolute pathname and assumes the working directory (or a higher directory in the structure) without explicitly referring to it. A relative pathname is one that begins either with a file or directory name or with one or more less-than symbols.

If the pathname begins with a name (for example, DIR1>FILEA or FILEA), the elements so identified are immediately subordinate to the working directory.

If a relative pathname begins with a less-than symbol (for example, <FOSTER), the name following the less-than symbol identifies an element that is immediately subordinate not to the working directory, but to the directory to which the working
* directory is immediately subordinate. If the pathname began with two less-than symbols (for example, <<APP2), APP2 is immediately subordinate to a directory two levels higher than the working directory.

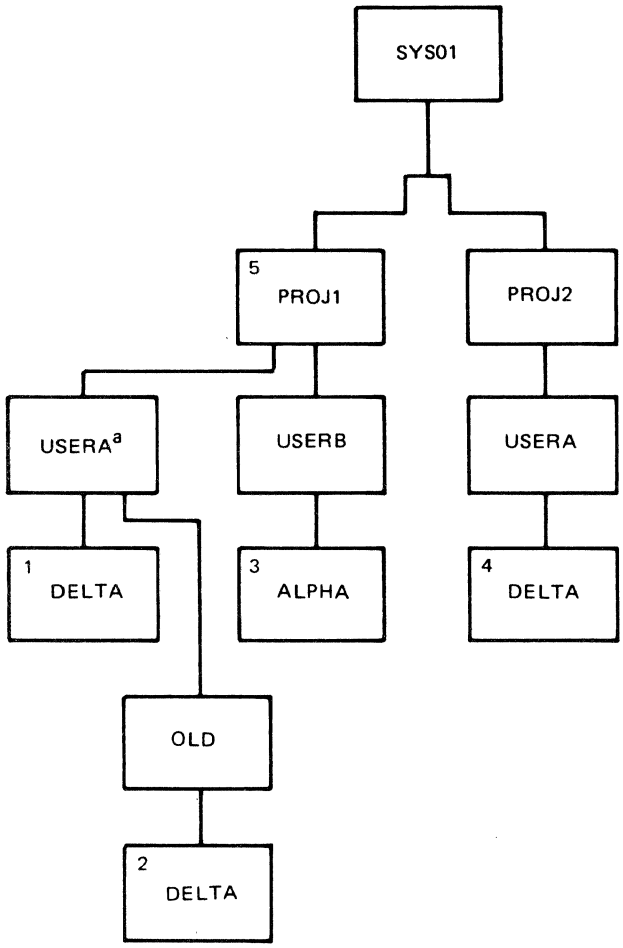
A relative pathname contains one or more names. If it contains more than one name, each name except the last must be a directory name, the first being immediately subordinate to the current working directory level (or to a higher level, as specified by one or more less-than symbols), the second immediately subordinate to the first, and so on. The last or only name can be a directory name or a file name, depending on the function being performed.

A simple pathname is a special case of the relative pathname. A simple pathname consists of only one name: the name of the desired element that is immediately subordinate to the working directory.

You can refer to a file or directory that is on the same volume (but not subordinate to the working directory) by using an absolute pathname or by using any of the described forms of a relative pathname.

Figure 2-3 shows some relative pathnames and the full pathnames they represent when the working directory pathname is >PROJ1>USERA.

BOX NUMBER	RELATIVE PATHNAME[a]	FULL PATHNAME
1	DELTA	^SYS01>PROJ1>USERA>DELTA
2	OLD>DELTA	^SYS01>PROJ1>USERA>OLD>DELTA
3	<USERB>ALPHA	^SYS01>PROJ1>USERB>ALPHA
4	<<PROJ2>USERA>DELTA	^SYS01>PROJ2>USERA>DELTA
5	<	^SYS01>PROJ1



[a]ASSUME CURRENT WORKING DIRECTORY IS ^SYS01>PROJ1>USERA.

84-819

Figure 2-3. Sample Pathnames

DISK DEVICE PATHNAME CONSTRUCTION

A special pathname convention is used to specify an entire disk volume. (This pathname convention is typically used in volume copy, create, and dump requests.) The special pathname consists of an exclamation point (!) followed by the symbolic device name and, optionally, the name of the the disk volume. The general form of the disk device pathname is:

```
!dev_name[>vol_id]
```

where dev_name is the symbolic device name defined for the disk device at system building, and vol_id is the File System name of the disk volume, without the circumflex (for example: !MSM00>VOL01).

If the vol_id is not supplied, reservation of the disk is exclusive (meaning that the reserving task group has read and write access but other users are not allowed to share the volume). This pathname form is used when a new volume is being created. If the vol_id is specified, reservation is read/share (meaning that the reserving task group has read access only, other users may read and write). This pathname form is used when copying a volume, or when dumping selected portions of a volume without regard for the hierarchical File System tree structure.

LINKS

Links are names you create through the Link Name command to refer to files, directories, and indexes in other volumes or directories as if they were in your working directory (or any other specified directory). Instead of copying a file from one directory to another, you can link to it. You can also link to devices or to other links.

For example, once you have established a link between the name A (in a given directory) and the pathname ^VOLID>MYDIR>MYFILE, you can perform file operations using the link-name A as if it were the pathname. Instead of having to issue the command:

```
MFA ^VOLID>MYDIR>MYFILE -RECOVER
```

you can issue the command:

```
MFA A -RECOVER
```

(Assuming you have defined A as a link-name in your current working directory.)

For additional information concerning the Link Name and Unlink Name commands, refer to the Commands manual.

Automatic Disk Volume Recognition

The automatic volume recognition facility dynamically notes the mounting of a disk volume. This feature allows the File System to record the root directory name in a device table. All references to disk files and directories begin, explicitly or implicitly, with a root directory name; therefore, every mounted file is automatically accessible to the File System software.

Disk File Organization

Since no one disk file organization can meet the needs of all users at all times, MOD 400 supports several different organizations, each of which is well suited to a particular application. Most of the supported organizations are based on the concept of a control interval (a unit of transfer between memory and disk) and are referred to as Unified File Access System (UFAS) files. UFAS file organizations provide file processing compatibility across the GCOS Executives.

You establish the organization of a data file when you create the file through the Create File command. You read and write the file using statements and macrocalls provided by the MOD 400 compilers and Assembler.

The following paragraphs summarize the MOD 400 disk file organizations. Refer to the Data File Organizations and Formats manual for detailed descriptions of each organization.

UFAS SEQUENTIAL DISK FILE ORGANIZATION

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 through the use of a simple key. Records can be read, modified, or deleted directly when you specify their exact control interval and record address (simple key). Records cannot be inserted; they can be appended to the end of a file. Fixed- or variable-length records can be used. If a record is deleted, the position it occupied cannot be reused.

UFAS RELATIVE DISK FILE ORGANIZATION

A 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). Both sequential and direct access are supported; in direct access, simple and relative keys can be used. A record can be updated (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 if you specify relative record numbers in random sequence.

UFAS INDEXED DISK FILE ORGANIZATION

In an indexed disk file organization each logical record contains a fixed-size key field that occupies a fixed position. Records are logically ordered by key value; they can be accessed sequentially in key sequence or directly by key value. Fixed- or variable-length records can be used. Variable-length records occupy variable-length slots. 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 sequence by key value.

UFAS RANDOM DISK FILE ORGANIZATION

In a random disk file organization records are accessed directly or sequentially. Variable-length records occupy variable-length slots. Direct access of records is performed through CALC keys, which are fixed in size and located within each record. Records are positioned according to a technique involving an arithmetic derivation of their CALC keys. This derivation is called a hashing algorithm (and is carried out by the system). Insertions, updates, and deletions are handled according to key value. When the file is initially loaded, records can be supplied in random key value sequence.

UFAS DYNAMIC DISK FILE ORGANIZATION

A dynamic disk file can contain fixed- or variable-length records and supports inventory information to describe available space. The main purpose of this file organization is to provide an efficient storage organization for records to be accessed through alternate indexes (explained below). Records are accessed sequentially or directly. Variable-length records occupy variable-length slots. Records can be accessed indirectly through alternate indexes or directly by specifying their exact control interval and record address (simple key). Records are inserted into the file according to inventory information on a "best fit" basis. When the file is initially loaded, records can be supplied in random key value sequence.

NON-UFAS RELATIVE DISK FILE ORGANIZATIONS

Non-UFAS relative disk file organizations are specific to the DPS 6 and are not compatible with other GCOS systems. These file organizations have fewer functional capabilities than UFAS files but require little or no space overhead. The non-UFAS file organizations are fixed relative and string relative.

- Fixed relative - A fixed relative disk file can contain only fixed-length records. All records in the file are considered active; the file cannot contain deletable records. A fixed relative file can be accessed directly or sequentially. New records can be inserted anywhere in the file.

- String relative - A string relative disk file can contain variable-length records. All records in the file are considered active. A string relative file can be accessed directly or sequentially. The ASCII line feed character (0A) is automatically appended to the end of each record.

Pipes

A pipe is a special kind of UFAS sequential file that is used for synchronizing and passing information among multiple cooperating tasks. Pipes are accessed (reserved, opened, read, written, closed, and removed) just like any other sequential file. Pipes provide a synchronization/queuing facility and offer a convenient way of organizing and distributing work.

One or more tasks write into the pipe while others read from it. If the pipe is empty but open for writing, read requests are suspended until data (a logical record) is available. A read implicitly deletes the logical record just read from the pipe. When the pipe is empty and no longer open for writing, read actions return the normal end-of-file status.

Alternate Indexes

Alternate indexes allow you to define any number of alternate record keys to provide any number of different logical orderings of keyed records within a single disk file. In effect, alternate indexes provide different orderings (views) of the same data. The same data file can be viewed in many different ways by having more than one alternate index. For example, an application could have a UFAS relative file containing employee information with alternate indexes for employee numbers, employee names, and social security numbers. You could access such a file as a relative file, as an indexed file ordered by employee numbers, as an indexed file ordered by employee names, or as an indexed file ordered by social security numbers.

The alternate index capability exists in addition to the normal access mode based on type of file. You can establish an alternate index for any UFAS relative, indexed, random, or dynamic disk file. A file with more than one index can be accessed in a number of ways. The manner in which the file is reserved (through the Get File command) determines how the file is accessed. If the data file itself is reserved, the file can be accessed normally (according to file organization) or by a key that is supported by one of the indexes. When the data file is reserved through an alternate index, the contents of the file can be accessed as a standard indexed file. Additionally, if more than one index exists, the indexes can be used as alternate keys to refer to the data. When an alternate index is used for file reservation, that index is used as the primary key and the remaining indexes can be used as alternate keys. Any index can be selected as a primary index. When one index is used to access the file, it and the other indexes are automatically updated as the file is updated.

UFAS dynamic disk files contain inventory information to manage available file space. Therefore, in highly volatile file environments that include many insert and delete operations, dynamic disk files are the ideal data files to be used with alternate indexes.

Character string, signed binary, signed unpacked decimal, and signed or unsigned packed decimal key types can be used. Single component keys, ordered in ascending or descending sequence, are supported. Duplicate keys (more than one record in a file with the same key value) are supported on an index-by-index basis.

An alternate index is created with the Create Index command. Arguments of this command specify the name of the index and the name of the data file with which it is to be associated. The system creates the index on the same directory as the data file and, unless otherwise specified, with the same control interval size as that of the data file.

Refer to the Data File Organizations and Formats manual for further information.

Disk File Protection

The File System provides facilities that enable you to control the access to files and directories, to control the concurrent access to files, and to control the contention for records within shared files.

ACCESS CONTROL

Access control is an optional File System feature that allows the creator of a file or directory to specify which users (if any) are to be granted access to the file or directory and what types of access these users are to be granted.

There are two general forms of access control: Access Control Lists (ACLs) and Common Access Control Lists (CACLs). ACLs apply directly to a file or directory; CACLs apply equally to all immediately subordinate entries in a directory. Entries in the ACLs and CACLs are managed through Set Access, Delete Access, and List Access commands.

Access control is a file or directory attribute. The File System maintains in each directory a list of users and the type of access each user is allowed. If a directory does not contain such a list, the items contained within it are not protected and are accessible to all users. (Access control applies only to disk files and directories. Tape files and other device-type files such as terminals and card readers cannot be protected through the access control facility.)

Access Types

Access types for files are read, write, and execute. Access types for directories are list, modify, and create. A null access type applies to both files and directories. Null access indicates that no access is to be granted.

Access Control/User Id Relationship

Access control assumes that access to the system is controlled by a login process in which every user has a unique user id. This user id is composed of three elements that are specified at login and that remain unchanged during the time the user is logged in. The three elements are:

person.account.mode

person - Name of individual who may access the system.

account - Name of account to which work is charged.

mode - Further identification of the user (optional). Can name the mode in which the user is working (for example, interactive, absentee, or operator).

The elements of the user id can consist only of the ASCII uppercase and lowercase alphabetic characters (A-Z, a-z), digits (0-9), underscores (_), dollar signs ($), apostrophes (') and the uppercase and lowercase graphics whose hexadecimal equivalents are C0-FE (extended character set). Apostrophes and the characters whose hexadecimal equivalents are C0-FE can be used only in the person and account elements. For both the primary and extended character sets, uppercase and lowercase characters are equivalent (for example, JOHN.MOD400.AB is the same user id as JohN.moD400.ab).

The elements are separated with periods (.). When referencing user ids, you can replace any or all elements by asterisks (*); for example:

```
*.account.mode
person.account.*
*.*.*
```

When an asterisk appears in an element position, it is interpreted to mean any value that may exist. No test is performed to match this element of the user id. For example, if two persons (Smith and Jones) are registered in an account named FILE_SYS, the user id *.FILE_SYS.* matches either person in any possible mode. (The user id *.FILE_SYS.* matches all individuals registered to use FILE_SYS in any mode.)

Access Control Lists

There are four kinds of access control lists: file ACLs, directory ACLs, file CACLs, and directory CACLs.

- File ACL - A file ACL is a type of access control list that applies to a specific file and is considered to be a file attribute. It contains a list of those users who can access the file and their specific access rights (read, write, execute).

- Directory ACL - A directory ACL is a type of access control list that applies to a specific directory and is considered to be a directory attribute. It contains a list of those users who can access the directory and their specific access rights (list, modify, create).

- File CACL - A file CACL is a type of access control list that applies to all files immediately subordinate to a directory. A file CACL is considered to be a directory attribute that applies only to files contained in that directory. A file CACL contains a list of file users and their specific access rights (read, write, execute). Use of file CACLs can save disk space and search time if all or most files in a directory have the same access requirements. A file CACL does not override individual file ACLs set on files in the directory.

- Directory CACL - A directory CACL is a type of access control list that applies to all directories immediately subordinate to a directory. A directory CACL is considered to be a directory attribute that applies only to immediately subordinate directories. A directory CACL contains a list of directory users and their specific access rights (list, modify, create). Use of directory CACLs can save disk space and search time when all or most subdirectories have the same access requirements. A directory CACL does not override individual directory ACLs set on the subdirectories.

The Create Directory command allows a directory CACL to be established as a global directory attribute. The directory CACL is automatically passed down to subsequently created subordinate directories.

Checking Access Rights

When you reserve a file (through the Get File command or system service macrocall), the File System checks your right to access that file. You are said to be on the access control list if your user id matches an entry on the ACL or CACL in any of the forms noted below.

Universal access (no access restriction) is implied if neither an ACL nor a CACL exists for the file being reserved. If either list is present, it is scanned by access control.

The checking priority is ACL first, CACL second. If a match is found in the ACL for a fully specified user id (all three elements explicitly stated), the CACL is not inspected. If a match is found on a partially specified user id (one or more elements specified as an asterisk), the CACL is inspected for a more explicitly stated user id. The following list indicates the inspection hierarchy of user id formats in order of decreasing priority. For example, if you are granted access by an ACL entry in format 3, you can be denied access only by an ACL or CACL entry in format 1 or 2.

1. person.account.mode
2. person.account.*
3. person.*.mode
4. person.*.*
5. *.account.mode
6. *.account.*
7. *.*.mode
8. *.*.*

Access is checked only for the target file or directory; the access rights set on directories that may be traversed in reaching the target file are not checked. You may be denied access at some intermediate directory level and still gain access to a subordinate directory or file.

Access control lists do not prevent the system operator from accessing files and directories. It is suggested that physical access to the operator terminal be restricted.

FILE CONCURRENCY CONTROL

Concurrent read or write use of a file among task groups is established by the task group that first reserves the file. Concurrency control performs the following functions:

- Establishes how tasks in the reserving task group intend to access the file (read, write, or execute).
- Establishes what the reserving task group allows other task groups to do with the file.

If the file is already reserved, a task group's concurrency request (reservation) is denied if its intended access conflicts with the access permitted by a prior reserver. The concurrency request is also 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. 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 File command or system service macrocall. The possible combinations of access intended for the reserving task group and sharability permitted other task groups are given in Table 2-1. Table 2-1 also shows the Get File command arguments that establish the various concurrrencies.

Table 2-1. Disk File Concurrency Control

Reserving Task Group	Other Task Groups	Get File Arguments
Read only	Read only (read share)	-ACCESS R -SHARE R
	Read or write (read/write share)	-ACCESS R -SHARE W
Read or write	No read, no write (exclusive use)	-ACCESS W -SHARE N
	Read only (read share)	-ACCESS W -SHARE R
	Read or write (read/write share)	-ACCESS W -SHARE W

Compiler-generated programs, commands, sort operations, and other system software always request exclusive concurrency for files they reserve for users. Since the operator terminal must be reserved with read/write shared concurrency to allow concurrent access by many task groups, it cannot be specified as the path of the -COUT argument of a command that invokes a compiler.

The command-in, user-in, user-out, and error-out files are associated with the MOD 400 Command Processor (refer to "Command Processor" in Section 3). 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. You can use File Out commands to specify the concurrency with which the user-out and error-out files are to be reserved.

ACCESS CONTROL/CONCURRENCY CONTROL RELATIONSHIP

In an environment that employs access control, users must have certain minimum types of access privilege to obtain the specific type of concurrency control they specify in Get File commands or system service macrocalls.

Table 2-2 summarizes the relationship between access control and concurrency control for disk files, disk directories, and disk volumes. (Note that access control does not exist for other types of devices.)

Table 2-2. Access Control/Concurrency Control Relationship

Object	Desired Concurrency	Minimum Access
Disk files	Read Read/write	Read Read/write
Disk directories	Exclusive use Nonexclusive use	List/modify List
Disk volumes	Read or read/write	Modify access to root directory

SHARED FILE PROTECTION (RECORD LOCKING)

Record locking is a File System option that provides interference protection so that co-operating users can share and update file data. For example, with record locking in effect there can be many task groups running COBOL applications that read, write, and update record data in the same file or same set of files.

User applications often employ standard data management services to lock records as they access them. The purpose of the locks is to prevent other users from simultaneously getting access to these records. If other users could access the records, they might get information that is only partially updated or, as a result of some programming decision or error condition, may soon be removed from the file. Also, if there were no locks, two users could update the same records at the same time. In this situation the second updater would inadvertently remove any modifications made by the first updater.

For reasons such as these, record locking is a necessary feature in most file sharing environments. Moreover, in many file sharing environments it is important that more than one lock be simultaneously maintained. For example, an "update" transaction to a parts inventory file may involve multiple record updates -- subtracting from some records and adding to others. These multiple record locks may even involve access to multiple files.

Note that record locking is not necessary to prevent a file from being physically corrupted by several applications performing multiple writes. Whether or not record locking is present, the File System maintains indexes and record chains properly so that the file structure is consistent. However, without record locking there is no synchronization, and the file data can be logically corrupted by two or more users who update the same data records. Also, without record locking, data can be viewed in a partially updated or inconsistent state.

Record Locking Implementation

The MOD 400 record locking option provides synchronization mechanisms to lock out record data as it is accessed, thereby making the data inaccessible to other applications until it is explicitly unlocked via a cleanpoint call (a call to the ZCLEAN utility in higher-level languages, or $CLPNT macrocall in Assembly language).

The File System locks records by maintaining lists that describe which file control intervals are locked, who has them locked, and who is waiting for them to be unlocked. The File System also provides a mechanism to recognize (and signal) whenever a deadlock condition occurs. A typical example of a deadlock is when one user owns (has locked) record A and wants to lock record B while another user already has record B locked and is waiting for record A to be unlocked.

When record locking is in effect and records are accessed through standard read-, write-, rewrite-, and delete-record calls, records are automatically locked for reading or writing until explicitly unlocked through a cleanpoint call. Record locking is performed on a shared-read/exclusive-write basis, which allows many simultaneous readers but only one writer at a time. This convention means that readers will wait until a writer finishes (issues a cleanpoint call) and vice versa.

Normally a reader is determined at open time as a user who has opened the file for read access only, while a writer is a user who has opened the file for write or update access. However, if a file has been designated as recoverable (see "File Recovery" in Section 6), the determination of reader and writer is made at each data access request. Read-record operations set read locks; write-record, rewrite-record, and delete-record operations set write locks.

NOTE

Since, with recoverable files, the type of lock is determined dynamically at each access request rather than once at open time, more readers can access more file data at the same time. Designating a file as recoverable will improve performance if requests commonly involve reading or searching through large amounts of data. Also, if two or more readers attempt to update the same data in a recoverable file, the data can be rolled back and the program restarted.

Setting Record Locking

You can set record locking as a permanent file attribute when you create the file or modify its attributes. You can set record locking temporarily (only while the file is reserved) when you reserve the file for processing.

To set record locking as a permanent file attribute, you specify the -LOCK argument of the Create File or Modify File commands. To set record locking temporarily, you specify the -LOCK argument of the Get File command. To change from locking to no locking, you specify the -NOLOCK argument in any of these commands.

A file having the record locking attribute can be reserved without record locking through the -NOLOCK argument of the Get File command. This is a special "dirty reader" option that lets you read data even though the data may be locked or may be in the process of being updated by some other user. The consistency and integrity of any data read is not guaranteed. The "dirty reader" option is available only on a logical file number (LFN) basis. The associated LFN is read-only, ignores any existing record locks, and cannot set any locks. (LFNs are internal file identifiers associated with file pathnames at the command or source program level; refer to "Logical File Numbers" in Section 5 for further information.)

A file with the record locking attribute can also be reserved with the no-wait option through the -NOWAIT argument of the Get File command. If the no-wait option is specified, the File System returns an error status rather than causing you to wait for a record to be unlocked. The no-wait option is available only on an LFN basis.

Record Locking Considerations

When using record locking, you should be aware of the following points:

- To efficiently use record locking, your applications must be written to be transaction-oriented so that records are not locked for a long period of time (for example, while waiting for terminal I/O) and so that as few records as possible are locked to satisfy the request.

- You should consider using other file integrity features (described in Section 6), especially file recovery, which allows data to be reset (rolled back) to the state it was in at the start of the transaction. In many situations, file recovery is a necessary feature to maintain data integrity in the event of system failure, record deadlock, application failure, terminal failure, and so forth.

- To develop an efficient multiuser application that shares and updates data in standard files, you must examine where and how the application is accessing file data and design the file structure carefully. In addition, you must pay careful attention to error conditions involving data recovery and program or transaction restarts.

- Applications that receive a deadlock notification must be prepared to back out of the current "transaction" and free up the locks they concurrrently own. If the file is recoverable, this is done through the rollback call (a call to the ZCROLL utility in higher-level languages; a $ROLBK macrocall in Assembly language).

- When a record is locked, the entire control interval in which the record is contained becomes locked. When defining control interval size, you should consider not only I/O transfer size and memory buffer usage, but also the number of records that may be locked out.

- If the no-wait option is selected, central processor time must be given up so that other users who have the record locked get a chance to unlock it. You may need to add a "suspend for time interval" function to applications using the no-wait option to allow other task groups enough time to finish their I/O and unlock records (issue a cleanpoint call).

- Closing a file, issuing a cleanpoint call, or issuing a rollback call frees up all records locked by the task group since the last cleanpoint. If a task group abort occurs, the system issues a rollback call automatically as part of the task group cleanup process. Likewise, if a system failure occurs, the operator will issue the Recover command after the system is restarted. When this command is issued, the File System (in effect) performs a rollback call for all task groups that were active at the time of the failure.

Remote File Access

Remote file access is a File System facility that allows applications to access remote data as if it were local. Remote objects such as files, volumes, magnetic tapes, and printers physically reside in some other computer node but, through remote file access, appear to be attached to your system. The remote file access facility captures references to remote objects and interfaces with the appropriate networking software (DSA for example) to get the desired function performed remotely.

When accessing data at another computer node, you may employ any File System function through macrocalls or higher-level language I/O statements. No special naming conventions are necessary. You supply the same kind of pathname you would to access local data. The File System checks to see if the object identified in the pathname is online (located on your node). If the object is not online, the File System checks a remote file catalog to see if the object is located at some other computer node.

The Remote File Catalog command is used to manage catalog information. This command allows a system operator or administrator to define, update, and display information about remote and local objects, nodes, and networking software.

The Remote File Access command is used to initiate the remote file access facility. This command allows the system operator or administrator to start the facility, retrieve network status information, and open and close connections between nodes.

REMOTE FILE CATALOG

Each node has its own remote file catalog to identify objects it can reference through remote file access. The catalog contains only those objects of interest to the node. A remote file catalog contains information about both remote and local objects.

Remote Object Information

Remote object information consists of a list of remote volumes and devices along with the node at which they are currently located. The File System allows you to define your own names for remote objects. For example, the line printer known as LPT04 in NODE3 can be cataloged as LPT01 in NODE1. Any reference to LPT01 in NODE1 will result in a search of the catalog and subsequent use of the printer through the remote file access services. The catalog can be updated dynamically by an operator or system administrator to configure new remote devices or to reconfigure existing ones.

Local Object Information

Local object information consists of a list of your volumes and devices that can be accessed from other nodes. This information is used by the other nodes to verify the existence of what is to them a remote object. It enables the File System to automatically update the catalog when volumes are moved from one system to another.

Volume Identification

Disk volumes can be exchanged or moved from one node to another. The remote file catalog contains enough information to uniquely identify a remote disk volume and to recognize when it has moved to another node. This information consists of:

- A Node of Birth (NOB) field identifying where the volume was originally cataloged for remote access.
- A Date of Birth (DOB) field identifying the date and time the volume was originally cataloged for remote access.
- A Node of Residence (NOR) field identifying the node where the volume is currently located (cataloged as a local object).
- A Node Migration Number (NMN) field identifying the number of times a volume has moved from one node to another.

The remote file catalog maintains the relationship between a local name, its current location (NOR), and the actual name. When a volume is moved to another node, only its NOR is changed, no change is made to the local name.

When a node connection is established, the two systems involved exchange local object information. A node ignores any volume information received that is out of date with respect to what it already has in its catalog. If a node has been off-line for some time, any old information it has will be discarded and any new information it received will be factored in.

Establishing Remote File Catalogs

Establishing a remote file catalog is usually a one-time operation. The steps involved in setting up the catalog are as follows:

1. Create the catalog.

   You create the catalog by using the Remote File Catalog (RFC) command with the -CAT argument. This step is performed only once.

2. Catalog a node for remote access.

   You define the nodes with which you are to communicate by using the RFC command with the -NODE argument. This step is repeated once for each node.

3. Catalog a local object to be accessed remotely.

   You define a local object that is to be accessed from other nodes by using the RFC command followed by the local pathname of the object. This step is repeated once for each local object to be addressed remotely. Any device to be cataloged must be configured on your system. A disk volume to be cataloged must be mounted.

4. Enable the remote file access facility.

   You invoke the remote file access facility by using the Remote File Access (RFA) command with the -STARTUP argument. This step configures and initializes the facility for communicating between nodes. It must be performed at the local node and each remote node whose objects are to be cataloged.

5. Establish communication with a remote node.

   You establish communication with the remote node by using the RFA command with the -OPEN argument. This step must also be performed at each remote node whose objects are to be cataloged.

6. Catalog a remote object to be accessed locally.

   You define a remote object that is to be accessed from your node by using the RFC command followed by the local name to be used to reference the object and the name of the node at which the object is located. If you wish to define a local name that is different from the name of the object as it is known at the remote node, you must use the -ROBJ argument. This step is repeated once for each remote object to be addressed locally. Note that communication must have been established with the remote node through the RFA command.

INITIATING REMOTE FILE ACCESS OPERATIONS

Once a remote file catalog is set up, only two steps must be performed on a day-to-day basis before you can access remote devices and data.

1. Enable the remote file access facility.

   You and the nodes with which you are to communicate must issue the RFA command with the -STARTUP argument.

2. Open the remote nodes.

   You must issue the RFA command with the -OPEN argument for each node with which you are to communicate. Those nodes that are to access objects at your node must also issue the RFA command with the -OPEN argument.

After you have enabled the remote access facility and opened the remote nodes, you can perform any operation on the remote data that you would perform on local data. Whether you use MOD 400 commands or your own application programs, the data will appear to be located at your node.

REMOTE FILE ACCESS SECURITY

The following paragraphs describe the way in which the remote file access facility handles access control, record locking, and data commitment.

Access Control Lists

Access control lists define which users have access to data and what kind of access they have. When files are accessed remotely, the same level of file protection exists as when files are accessed locally. If a file is protected by an access control list, no local or remote user can access the file unless the user is given permission through the access control list.

Record Locking

Record locking prevents other users from simultaneously getting access to records that you are accessing. In many applications record locking involves multiple record locks on multiple files and, in networking environments, may involve locks to multiple files in multiple nodes.

A typical deadlock condition can occur if one user has locked some records and is trying to lock others while another user has these other records locked and is trying to lock the records already locked by the first user. The File System on your computer node knows about the record locks on local data files and is able to detect deadlocks. Since the File System does not know about record locks on remote data files, it prevents deadlocks from occurring by using a time stamp algorithm.

Users are assigned time stamps when they start to access remote data. The time stamps are passed to remote nodes by the remote file access facility. At a remote node, a user may only wait for records that are held by younger users (users whose time stamp is later). If the application attempts to lock a record that is already locked by an older user (a user whose time stamp is earlier), it receives a "deadlock has occurred" return status. The application must then abort, backtrack, or restart. If a record is held by a user who is local to that node, the local user is always considered the younger.

Data Commitment

For purposes of data integrity, an application that accesses and updates remote (and local) data may be structured in phases known as commitment units. The end of a commitment unit is a point at which the user is willing to commit changes to the data base. This type of application is said to be transaction oriented. It may complete successfully (commit) at a program-defined commitment point or it may fail (abort), in which case any updated data must be returned to its initial pre-transaction state. To ensure reliability, the transaction must either complete in its entirety or not complete at all.

In remote file access, data commitments are performed in two phases: precommit and commit.

- Precommit - All data is recorded on disk with an indicator to show that the data is in a precommitted state. This step is done locally. Remote file access then sends messages to precommit data at all remote nodes.

- Commit - Once messages have been received from all affected remote nodes indicating that all data is in the precommit state, local data is committed (unlocked and made available to other users). Another round of messages is then sent via the remote file access facility to commit data in the remote nodes.

If a system or node failure occurs in any intermediate step, there is enough information available so that, on restart, a decision can be made to commit or recover the data. More detailed information on file recovery is presented in Section 6.

Multivolume Disk Files

In most applications a disk file resides on a single volume. However, there may be situations in which you want to extend a file over more than one physical volume. The need for multivolume files could arise from any of the following:

- You want an endless sequential file capability similar to that available with magnetic tape.
- You want to define a single file that is too large to be contained on one volume.
- You want to improve access time to a file by spreading the file data over several volumes and/or separating the index portion of an indexed file from the data portion and placing the portions on separate volumes.

A multivolume file is treated as a collection of file sections. A file section is that part of the file that is contained on one volume. A file set is all of the sections making up the multivolume file.

MULTIVOLUME SETS

A multivolume set is a disk file that resides on more than one volume. A volume is identified as being part of a multivolume set when the volume is created through the Create Volume command.

Each multivolume set has a root volume (in which the set begins) and a number of additional volumes. All volumes that are part of the set are called members.

The name of a multivolume set is independent of the names of
* the volumes it contains. A volume is established as a member of a set when the set name and a sequential member number are specified at volume creation. The root volume is always member number 1.

There are two types of multivolume sets: online and serial. Online multivolume sets are used for all nonsequential multivolume files. They may also be used for sequential mutivolume files. Serial multivolume sets are an alternative for large sequential files. They are also used for files that require an endless sequential capability similar to that of magnetic tape.

The types of multivolume sets and files are described in detail below.

Online Multivolume Set

A volume is designated as part of an online multivolume set at volume creation. An online multivolume set has the following characteristics:

- All members of the set must be mounted and available while the set is in use.
- Member volumes, other than the root volume, can be used independently of other members in the set to contain single-volume files and directories.

Online Multivolume File

A file is designated an online multivolume file when it is created under a directory in the root volume of an online multivolume set. An online multivolume file has the following characteristics:

- Can have any UFAS file organization.
- Can be located by any type of pathname.
- Can skip set members when continuing to another volume.

Figure 2-4 illustrates the combination of files and volumes used by a sample online multivolume set. Multivolume files FILEA, FILEB, and FILEC must begin on VOL1. FILEX, FILEY, and FILEZ are single-volume files because they do not begin on VOL1. The pathnames used to access the files are shown at the bottom of the figure.

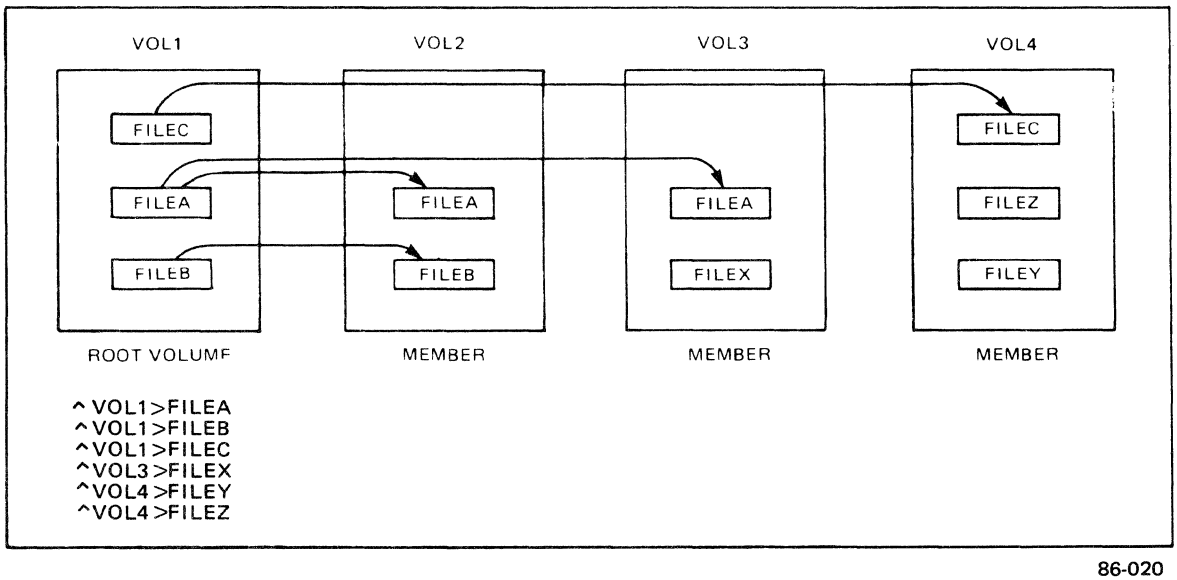


Figure 2-4. Example of Online Multivolume Files

Serial Multivolume Set

A volume is designated as part of a serial multivolume set at volume creation. A serial multivolume set has the following characteristics:

- No member of the set need be mounted until the data on it is required for processing.
- Any member of the set, including the root volume, can be used independently of other members of the set to contain single-volume files and directories.

Serial Multivolume File

A file is designated as a serial multivolume file when it is created in the root directory of a volume in a serial multivolume set. A serial multivolume file has the following characteristics:

- Must be a UFAS sequential file.
- Must be cataloged in the root directory of the volume on which it starts. More than one serial multivolume file can belong to a set, and each such file can begin on a different volume if desired.
- Must be located through a pathname of the form ^volid>filename.
- Must continue serially from one volume to the next.

Figure 2-5 illustrates the combination of files and volumes used in a sample serial multivolume set. Serial multivolume file A begins in VOL1. Serial multivolume file B begins in VOL2. Both continue in other volumes of the set. Files C, D, and E are single-volume files. The pathnames by which the files are located are shown at the bottom of the figure.

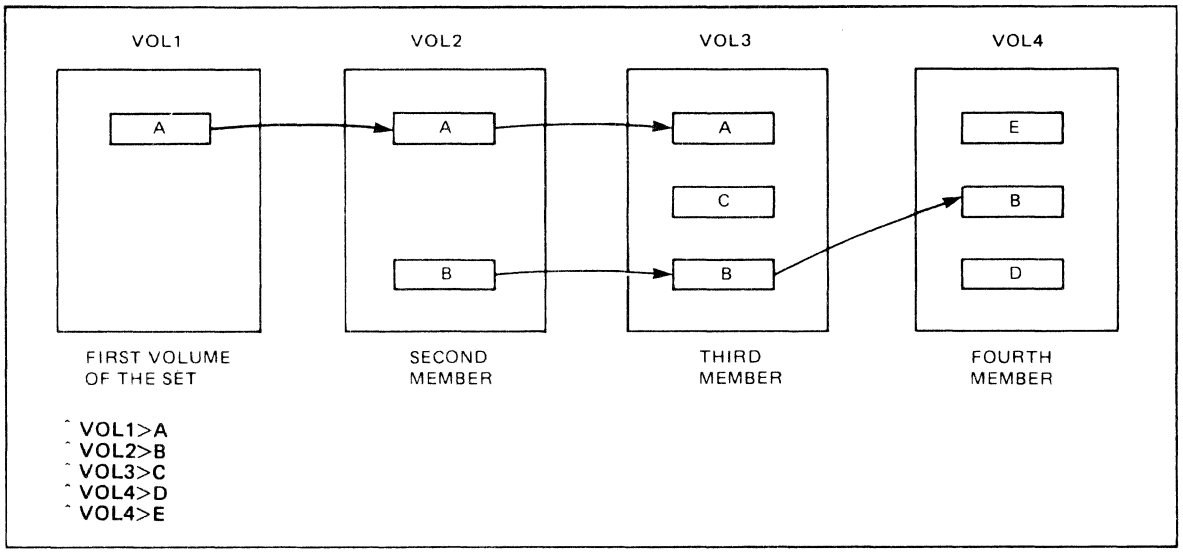


Figure 2-5. Example of Serial Multivolume Files

Disk File Buffering

A buffer is a memory storage area used to compensate for a difference in the rate of data flow, or time of occurrence of events, during transmission of data from one device to another. Buffering is the process of allocating and scheduling the use of buffers. In some applications, overlap of input operations and processing can be achieved by anticipatory buffering, where the next block of data 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.

FILE ACCESS LEVELS

Disk files can be processed at either block or record level. In block level access, data is transferred directly between the file and a buffer in your program. Your program must perform all buffer management operations. In record level access, the system assigns disk files to buffer pools when your program opens the files. The system buffering facilities are used to perform all buffer management operations.

BUFFER POOLS

When a file is opened for read, write, rewrite, or delete operations, the File System assigns the file to a particular buffer pool. A buffer pool is a collection of buffers that provides a method of conserving memory for disk file access. Buffer pools are designed to:

- Reduce the amount of memory required for buffers by all users.

- Reduce the number of I/O operations in a random access environment.

- Provide more flexibility for shared file applications.

All buffers in a pool are the same size. Any number of files with matching control interval sizes can be assigned to the same buffer pool. A particular file, however, can be assigned to only one pool.

Each buffer in a buffer pool can store a disk control interval. When an application program issues a read instruction and the desired record is not in any buffer, the next empty buffer is filled with the control interval containing the record. When all buffers are filled, an active buffer is selected for the next different control interval according to a least-recent-usage algorithm.

In addition to conserving memory when disk files are accessed, buffers eliminate the need for each user to define private buffer areas. One or more system-wide buffer pools should be created at system startup (through a startup EC file; see Section 3). Users who have special buffering requirements can create their own buffer pools for files they reserve exclusively.

Types of Buffer Pools

Each buffer pool is created as either a public or a private buffer pool, and can be considered file-specific or general. Buffer pools are created by the Create Buffer Pool command and deleted by the Delete Buffer Pool command. When creating a buffer pool, you specify the number of buffers it is to contain, the buffer size, and (optionally) the name of the buffer pool.

- Public buffer pools - Public buffer pools are those created by the operator or the system startup EC file. Public buffer pools reside in system memory and are available to all files and task groups. A disk file is assigned to a public pool if its control interval size (specified in the command that creates the file) matches the pool's buffer size.

  In many environments, three or four public buffer pools corresponding to three or four common file control interval sizes are sufficient for all performance and buffering needs.

If the system volume is associated with the disk cache processor, heavily used directories, as well as files that are read sequentially, are likely to be resident in the disk cache buffer. Buffer pools for these directories and files may not be needed.

- Private buffer pools - Private buffer pools can be created by each user. Private buffer pools reside in the task group's memory space and are available only for disk files reserved exclusively by that task group. A disk file is assigned to a private pool if the file is reserved for exclusive use and its control interval size (specified in the command that creates the file) matches the pool's buffer size. Private buffer pools should be created only if necessary to meet specific buffering needs. Public buffer pools should be sufficient in most cases.

- File-specific buffer pools - When you reserve a disk file with the Get File command, you can specify the number of buffers (using the -NBF argument) to be used when accessing the file. When the file is opened, a buffer pool is automatically created for use only by that file. This file specific pool is created in the task group's memory if the file is reserved exclusively, or in system memory if the file is reserved as sharable. The -NBF argument should be used carefully since it prevents a file from being assigned to a public or private buffer pool.

Buffer Pool Optimization

The File System collects a set of statistics on the use of each buffer pool. The installation can use this information to optimize disk I/O operations. Buffer pool statistics are obtained through the Buffer Pool Status and Buffer Pool Information commands. The Buffer Pool Status command provides a summary of the public or private buffer pool status. The Buffer Pool Information command provides a detailed status report on a particular buffer pool.

The installation should analyze applications and their associated file usage to fully utilize the advantages offered by buffer pools. Only a limited number of control interval sizes should be allowed for user files. In general, buffer and control interval sizes should be chosen to evenly distribute high and low activity files over the various buffer pools, thus reducing the amount of contention in the pools. The initial determinations will provide an acceptable level of performance and provide the basis for further analysis.

The Adjust Buffer Pool command can be used to temporarily alter the number of buffers in a private buffer pool. Once the most efficient buffer pool size has been established, it should be permanently fixed through the Create Buffer Pool and Delete Buffer Pool commands.

MAGNETIC TAPE FILE CONVENTIONS

The magnetic tape file conventions discussed in the following paragraphs include file organization, naming conventions, pathnames, and buffering operations.

Tape File Organization

The following information applies only to 1/2-inch, 9-track magnetic tapes.

Magnetic tape supports only the sequential file organization. Fixed- or variable-length records can 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.

The unit of transfer between memory and a tape file is a block. Block size varies depending on the number of records and whether the records are fixed or variable in length.

A block can be treated as one logical record called an "undefined" record. An undefined record is read or written without being blocked, unblocked, or otherwise altered by data management. Spanned records (those that span across two or more blocks) are supported. No record positioning is allowed with spanned records.

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

- Single-volume, single-file
- Multivolume, single-file
- Single-volume, multifile
- Multivolume, multifile.

The following types of unlabeled tapes are supported:

- Single-volume, single-file
- Single-volume, multifile.

Magnetic Tape File and Volume Names

Each tape file and volume name in the File System can consist of the following ASCII characters: Uppercase alphabetics (A through Z), lowercase alphabetics (a through z), digits (0 through 9), exclamation point (!), double quotation marks ("), dollar sign ($), percent sign (%), ampersand (&), apostrophe ('), left parenthesis ((), right parenthesis ()), asterisk (*), plus sign (+), comma (,), hyphen (-), period (.), slash (/), colon (:), semicolon (;), less-than sign (<), equal sign (=), question mark (?), underscore (_), and the characters whose hexadecimal values are in the range C0 through FE (extended character set).

NOTE

If the terminal is not capable of processing 8-bit data, characters from the extended character set are displayed as periods or as their 7-bit equivalents.

The underscore character can be used as a substitute for a space. If a lowercase alphabetic character is used, it is converted to its uppercase counterpart ("DATA', "Data" and "data" all refer to the same file).

Any of the characters defined above can be used as the first character of a file or volume name.

The name of a tape volume can be from 1 through 6 characters in length. Tape file names can be from 1 through 17 characters.

Magnetic Tape Device Pathname Construction

A magnetic tape volume must be dedicated to a single user. For this reason, the device pathname convention must always be used when referring to magnetic tape volumes or files. The general form of a tape device file pathname is:

```
!dev_name[>vol_id[>filename]]
```

where dev_name is the symbolic name defined for the tape device at system building, vol_id is the name of the tape volume, and filename is the name of the file on the volume. Tape devices are always reserved for exclusive use (the reserving task group has read and write access, other users are not allowed to share the file).

Automatic Tape Volume Recognition

Automatic volume recognition dynamically notes the mounting of a tape volume. This feature allows the File System to record the volume identification in a device table, thus making every tape volume accessible to the File System software.

Magnetic Tape Buffering

The -NBF argument of the Get File command can be used with magnetic tape files to reserve one or two buffers. If -NBF is not used, the File System attempts to allocate two buffers. If two buffers are allocated, the File System does "double buffering." When the tape file is being read, the File System unblocks one buffer while an anticipatory read is done into the other buffer. Similarly, when the tape file is being written, the File System blocks records into one buffer while a previously filled block is written out of the other buffer. This allows application code to execute in parallel with I/O transfers.

UNIT RECORD DEVICE FILE CONVENTIONS

Unit record devices (card readers, card punches, printers, terminals, and paper tape reader/punches) are used only for reading and writing data. They are not used for storing data, and thus do not require conventions for file identification and location.

Unit Record Device Pathname Construction

The pathname of a unit record device consists of the symbolic device name defined at system building preceded by an exclamation point (!). The pathname format is:

```
!dev_name
```

where dev_name is the symbolic device name of the unit record device.

Unit Record Device Buffering

All printers and most interactive terminals are provided with one File System buffer. (The operator terminal cannot be buffered.) By providing a File System buffer, application code can execute in parallel with I/O transfers.

All printers and all terminals (except the operator terminal) have a tabbing capability through software that converts the tab into spaces. Default tabulation stops are set at position 11 and at every tenth position thereafter for the line length of the device.

UNIT RECORD READ OPERATIONS

When an application task issues a logical read to a File System buffered device, one of the following actions occurs:

- If the buffer is full from a prior anticipatory read, the data in the buffer is transferred into the application task's area and a physical I/O transfer (an anticipatory read) is performed in parallel with continued 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; for other read actions it is the same.

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.

Interactive Terminal

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 or by checking for the appropriate status return after a Read, the application will not be stalled if the terminal operator is not present at the time of the Read request. Instead, the application can continue to poll other terminals.

Immediately after the Open is complete, a physical connection is made while the application continues execution. Depending upon the language the application is written in (for example, FORTRAN or Assembly language), it may be able to check the status of the Open to see if a Read can be issued without stalling application execution. The File System issues an asynchronous anticipatory physical read when the status check following 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" (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 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 the application until the anticipatory read is complete.

To avoid stalling on a Read or to avoid status check looping to test the input buffer status, applications should put themselves in the wait state, thus making the central processor available for lower priority tasks.

After the Open, an application written in COBOL must issue Read requests. The application will be put in the wait state if it is executing I/O statements in synchronous mode. Otherwise, the COBOL run-time package performs status checks and returns a 9I status until successful completion. The program can either loop on the Read or continue other processing.

BUFFERED WRITE OPERATIONS

A buffered write operation to a unit record device works on behalf of the application program in the same logical manner as a read operation. 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 devices.

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 is moved into it again. The application can avoid stalling execution when writing to an interactive terminal by doing one of the following:

- Checking the status of the system buffer before issuing the Write to see if the interactive terminal is still in use.
- Testing for a particular status return after the Write.

If a Write call is issued while data is being entered into the system buffer (because of a Read), the following sequence of events takes place:

- The read is allowed to complete.
- Input data is saved in the system buffer.
- A synchronous write is reissued by the File System.
- Output data is transferred directly from the application buffer.

Note that tab characters are not expanded into spaces.

If a physical I/O error occurs while data is being transferred from the system buffer to the device, you must be aware that the error occurred on the previous write operation. Furthermore, if any type of error occurs, the application program may need to have saved (or be able to retrieve) the data record so that it can be repeated.

Section 3
SYSTEM ACCESS

You can request access to the system to perform a number of different functions, such as:

- System building - Configuring the system to the needs of its users.
- System administration - Registering users.
- Operation control - Starting up the system each day, controlling processing, managing peripheral devices, and monitoring system status.
- Program development - Compiling, testing, and debugging programs.
- Application execution - Interacting with a program to accomplish a particular task.

In a large installation, different individuals will perform different functions. In a small installation, one person may perform most or all of the functions.

Access to the system is restricted to authorized users by means of the registration and login processes. Access to system files is restricted to specified users through the access control process (described in Section 2). Access to the various system facilities is controlled through the menu or command environment.

Before any access to the system can be made, the system must first be configured.

SYSTEM CONFIGURATION AND DEFINITION

Creation of a system is a two-step procedure, consisting of:

1. Bootstrapping a Honeywell-supplied system startup routine that provides a limited operating environment for building the files used in the second step.

2. Specializing 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 application programs.

The bootstrap operation consists of turning on the power supply to the hardware, mounting the disk containing the MOD 400 system software, and pressing several keys on the control panel or System Control Facility device. (The procedure is described in the System User's Guide.) The bootstrap operation generates the initial configuration and startup operations. Procedures are executed to produce a one-user online environment that can be used to specialize system startup and develop or execute application programs. (On the DPS 6/22 these procedures produce an environment that can be used to invoke the Autoconfigurator.)

Once this limited environment has been created, you either invoke the Autoconfigurator (DPS 6/22 only) or you create a file (called the CLM_USER file) that containing the Configuration Load Manager (CLM) directives that describe the operating environment that will exist at your installation. The CLM_USER file is created automatically by the DPS 6/22 Autoconfigurator. The configuration directives are described in the System Building and Administration manual.

To further define the environment, you can modify the operator commands in the Honeywell-supplied START_UP.EC file that is immediately subordinate to the system root directory. (START_UP.EC files are described later in this section.)

After the CLM_USER file is created and the START_UP.EC file is modified, you again bootstrap the system. This time, the directives in the CLM_USER file control the configuration, and the operator commands in the system START_UP.EC file further define the operating environment.

USER REGISTRATION

User registration is a process that protects the system from unauthorized access. Each person who is to be allowed on the system must be registered by the system administrator. The administrator uses the Edit Profile command to specify user-specific information such as:

- User id (Identifies the user for system purposes. Consists of two or three parts: person.account[.mode]. Uppercase and lowercase characters are equivalent; JONES.ADMIN and Jones.Admin are the same user id.)

- Login id (Identifies the user for login purposes only. Contains no periods. Uppercase and lowercase characters are distinct; JONES and Jones are different login ids.)

- Default login line

- Login traits, including:

  - Single or multiuser profile
  - Secondary user profile
  - Login only with default login line

- Whether a password is required to log in

- Whether statistics such as the following are to be kept for the user:

  - Number of sessions
  - Total elapsed time
  - Accumulated system resource usage.

This information is stored in a profiles file. The profiles file contains the user profiles for all registered users. The Listener checks the user profile when monitoring the privileges and/or limitations of each user.

On the DPS 6/22, the Autoconfigurator creates a default profiles file that you can customize to meet your special requirements.

User profiles are made up of sections, each of which describes a specific interface to some part of the system. A user profile always contains a registration section (consisting of the information listed above plus the optional user statistics, if specified). It can contain the following optional sections:

- Comments section. Contains any comments the system administrator has about this user's registration.

- Subsystem sections. Contain user-specific information meaningful only to individual subsystems. Examples of subsystem sections are:

  - The Word Processing (WP) section, which contains information such as the printer assigned to the word processing user.

- The Menu Subsystem (MS) section, which contains information such as the name of the menu the user is to automatically see when he or she logs in.

The List Profile command allows registered system users to view the contents of their user profiles. It allows a system administrator to view any registered user's profile.

Refer to the System Building and Administration manual for details on user registration.

ACCESSING THE SYSTEM

Access to the system requires:

- A physical connection between your terminal and the central processor
- A logical connection between you and the Executive.

Connecting to the Central Processor

Two types of terminal connections are possible: a direct connection and a dialup connection. In a direct connection, the terminal is connected to the central processor when the terminal's POWER switch is turned ON and its ONLINE/OFFLINE switch is set to ONLINE. In a dialup connection, the terminal is connected to the central processor after a telephone number is dialed on the terminal's modem.

The same terminal can be used both as an operator terminal and as a user terminal.

Refer to the System User's Guide for further information on physical connections.

Connecting to the Executive

You can access a terminal in one of two ways, depending on whether or not the terminal is configured for login. Terminals configured for login are reserved for access to the system through a system component called the Listener. Such terminals cannot be directly reserved by system applications. Terminals not configured for login are not monitored by the Listener and can be directly reserved by system applications. (Refer to the System Building and Administration manual for details on configuring the Listener and Listener terminals.)

Systems are more secure when all terminals are monitored by the Listener.

LOGIN TERMINALS

When a terminal is monitored by the Listener, either a banner line or a login form is displayed when the connection to the central processor is made. You can connect to the Executive through one of the following login methods:

- Full login. Type the Login command and, if required, enter the designated password.
- Forms login. Fill out the login form and, if required, enter the designated password.
- Abbreviated login. Type the 1-character login abbreviation and, if required, enter the designated password.

Another login method, direct login, requires no action on your part beyond turning on the terminal.

The system builder defines those terminals that are to be monitored by the Listener through entries in a file called the terminals file. (On the DPS 6/22, the Autoconfigurator builds a terminals file that you can modify for special requirements.) The terminals file is used to specify the following information:

- Maximum number of concurrently logged-in users to be allowed on the system.
- Name of each terminal that can be used as a login terminal (plus, for direct login terminals, the login line to be used when the terminal is turned on.)
- Each 1-character login abbreviation and its associated login line.
- Other information about the system and specific terminals.

NON-LOGIN TERMINALS

An application can be activated with a terminal designated for the input of the commands and/or data required by the application. When this terminal is physically connected to the system, no banner or form is displayed. In most cases, you can start entering data immediately. The screen display and your response depend on the application.

A login terminal can be changed to a non-login terminal (and back again) by the Set Listen command.

Activated Lead Task

When you gain access to the system, the executable code for the lead task (the controlling task of the application) is loaded and activated. The lead task can be designated to be the Command Processor, the Menu Processor (part of the User Productivity Facility, or UPF), or an application. When the Command Processor is the lead task, you can control execution by issuing any user command described in the Commands manual. When the Menu Processor is the lead task, you can control execution by filling out screen forms. When an application is the lead task, neither the Command Processor nor the Menu Processor is part of the task group.

MENU ENVIRONMENT (UPF)

The UPF (also referred to as the menu subsystem) provides an easy-to-use interface to MOD 400. Instead of typing command lines, you fill in Honeywell-supplied online menus and forms. You select system commands by choosing options listed on these menus and then filling in the fields on one or more forms. When all appropriate fields have been filled in, the command is generated and executed.

The UPF has a menu that allows you to enter any line-oriented command on the screen.

Menu Processor

The Menu Processor is the system software component that reads the Honeywell-supplied menus and forms you fill in. After reading a form (or command line), the Menu Processor loads and executes a bound unit that fulfills the request represented by the form.

The essential parts of the menu environment are the Menu Processor and the command-in file. Three other files are involved with, but not limited to, the menu environment. These are the user-in file, the user-out file, and the error-out file. The files associated with the Menu Processor are also referred to as the standard I/O files.

COMMAND-IN FILE

The command-in file is the file on which the Honeywell-supplied forms are filled in and from which the generated or directly specified command lines are read.

USER-IN FILE

The user-in file is the file from which a command, 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 the subsequent command, the user-in file remains the same as the command-in file. At the termination of a command that names an alternate user-in file, the user-in file reverts to its initial assignment.

For example, the directives submitted to the line editor following the entry of the Editor command are submitted through the user-in file. No specific action is required on your part to activate or connect to the user-in file unless the directives are to be read from a previously created disk file. You simply invoke the line editor and begin entering line editor directives through the same terminal. The attaching of the terminal to the user-in file is invisible to you.

USER-OUT FILE

The user-out file is the file to which a task group normally writes its output. However, certain system components (for example, compilers) also write to list files (path.L) or to the output file defined in the -COUT argument of the command that invokes the compiler. The user-out file is initially established by the -OUT argument of the Enter Group Request or Spawn Group command. Originally, it is the same device as the error-out file device. It can be reassigned to a file or another device by use of the File Out command or New User Out system service macrocall. Such a reassignment remains in effect for the task group until another reassignment occurs.

Again using the line editor as an example, any responses from the line editor, such as the printing of a line of the file being edited, are issued through the user-out file. As in the case of the user-in file, you need not perform any special action to attach your terminal to the user-out file. The only time such action would be required is if you wanted to direct the output from the command to some device other than the terminal.

ERROR-OUT FILE

The error-out file is used by the system to communicate any error condition that may be detected during the filling out of a form or during command execution. Such a condition could be an incorrectly specified field (reported by the Menu Processor) or a file-not-found condition (reported by the generated command). The error-out file is originally the same as the original user-out file; it can be reassigned to a file or another device by use of the File Out command.

Menu Level

When the system is in a state in which it is capable of displaying a menu, it is said to be at menu level.

ACHIEVING MENU LEVEL

The Listener can spawn task groups whose lead tasks are the Menu Processor. Note that the system is delivered with a Honeywell-supplied user task group ($H) whose lead task is the Command Processor (described later in this section).

The system indicates it is at menu level by displaying a menu.

When executing a function, you can return to menu level in any of the following ways:

- Normal termination. At normal termination of a generated command function, the task group returns to the previous menu.

- Command interruption. You can interrupt the execution of an invoked command by pressing the Break key (which may be labeled with BREAK or BRK) on the terminal. The system responds to this action with the break message **BREAK** and displays the previous menu.

MENU LEVEL PROCESSING

When you fill out a Honeywell-supplied form and press the Execute key, the system performs the following steps:

1. Spawns a task naming the requested bound unit (the bound unit named by the generated command). Task spawning implies task creation, which consists of the allocation and initialization of any control structures and data areas required for task control.

2. Calls the Loader to load the requested bound unit.

3. Places a request for the execution of the bound unit against the created task. The Menu Processor enters the wait state to await completion of the requested task (command). At this point, the system leaves menu level, which can be returned to only by completion of command execution or by the pressing of the Break key.

4. If the command is Enter Group Request, places a group request against an application task group. The Enter Group Request command terminates. The request is queued if there are other outstanding requests against the application task group from previous Enter Group Request commands.

5. Deletes the spawned task when the command terminates. Optionally issues a ready message to indicate a return to menu level.

Menu Format

Each selection on a Honeywell-supplied menu and each form field that may be filled in has an associated help message. To obtain help in selecting an option on a menu or in filling in a field on a form, press the key designated as the help key for your terminal. When the help key is pressed, a help message is displayed below the menu or form. If you make a mistake in filling in a field, the system provides expanded error messages that list causes and corrective actions to aid in resolving the problem.

You can modify the UPF menus, forms, help messages, and error messages by using the same tools that Honeywell used to create them. You can add, delete, modify, and reword any of the UPF elements through the following tools:

- Menu Builder to create, modify, or delete menus
- Forms Developer (VISION) to create, modify, or delete forms
- Generalized Forms Processor (DFC) to process standard forms
- Add/Delete Message Utility to update a message library.
- Table Maintenance Utility to build or modify tables used to process forms.

For detailed information about using and maintaining UPF, refer to the Menu System User's Guide.

Subsystem Switcher

The UPF provides visibility to the Subsystem Switcher that allows a logged-in user to switch from one subsystem to another without having to log out and back in again. (A subsystem is an application-oriented facility that supports one primary user or multiple secondary users.) If you are registered as a subsystem switcher user, when you finish work under one subsystem and want to work under another, you exit the subsystem according to its rules. For example, if you were working in command mode, you would enter the Bye command. Rather than being logged out, you would be presented with the menu that was displayed when you logged in. You would then make a selection from that menu.

COMMAND ENVIRONMENT

The command environment is that environment in which you communicate with the Executive through command lines entered at a terminal or read from a command file.

Command Processor

The Command Processor is the system software component that reads your command lines. After reading a command line, the Command Processor loads and executes a bound unit that fulfills the request represented by the command line.

The essential parts of the command environment are the Command Processor and the command-in file. Three other files are involved with, but not limited to, the command environment. These are the user-in file, the user-out file, and the error-out file. The four files associated with the Command Processor are also referred to as the standard I/O files.

COMMAND-IN FILE

The command-in file 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 residing on a disk, as in the case of a non-interactive user.

USER-IN FILE

The user-in file is the file from which a command, 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 the subsequent command, the user-in file remains the same as the command-in file. At the termination of a command that names an alternate user-in file, the user-in file reverts to its initial assignment.

For example, the directives submitted to the line editor following the entry of the Editor command are submitted through the user-in file. No specific action is required on your part to activate or connect to the user-in file unless the directives are to be read from a previously created disk file. You simply invoke the line editor and begin entering line editor directives through the same terminal. The attaching of the terminal to the user-in file is invisible to you.

USER-OUT FILE

The user-out file is the file to which a task group normally writes its output. However, certain system components (for example, compilers) also write to list files (path.L) or to the output file defined in the -COUT argument of the command that invokes the compiler. The user-out file is initially established by the -OUT argument of the Enter Group Request or Spawn Group command. Originally, it is the same device as the error-out file device. It can be reassigned to a file or another device by use of the File Out command or New User Out system service macrocall. Such a reassignment remains in effect for the task group until another reassignment occurs.

Again using the line editor as an example, any responses from the line editor, such as the printing of a line of the file being edited, are issued through the user-out file. As in the case of the user-in file, you need not perform any special action to attach your terminal to the user-out file. The only time such action would be required is if you wanted to direct the output from the command to some device other than the terminal.

ERROR-OUT FILE

The error-out file is used by the system to communicate any error condition that 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-out file is originally the same as the original user-out file; it can be reassigned to a file or another device by use of the File Out command.

Command Level

When the system is in a state in which it is capable of accepting a command from the command-in file, it is said to be at command level.

ACHIEVING COMMAND LEVEL

You can achieve command level by creating or spawning a user task group whose lead task is the Command Processor. The Listener can also spawn task groups whose lead tasks are the Command Processor. The system is delivered with a Honeywell-supplied user task group ($H) whose lead task is the Command Processor.

The system indicates it is at command level by issuing a "ready" prompter message at your terminal. (This assumes that you have not disabled the ready message by issuing a Ready Off command. If Ready Off has been issued, the system comes to command level without informing you.) If you are working in the system task group ($S) at the operator terminal, no ready prompt message appears unless you issue an EC !CONSOLE command followed by a Ready On command.

When executing a command function, you can return to command level in any of the following ways:

- Normal command termination. At normal termination of a command function, the task group returns to command level and awaits the entry of another command. (If your terminal is not monitored by the Listener, you should not enter the Bye command.)

- Command interruption. You can interrupt the execution of an invoked command by pressing the Break key (which may be labeled with BREAK or BRK) on the terminal. The system responds to this action with the break message **BREAK**. At this point, you can enter other commands or can enter the Start command to resume processing at the point of interruption. (Refer to the Commands manual for details.)

COMMAND LEVEL PROCESSING

When a command such as Copy, Change Working Directory, or Enter Group Request is read by the Command Processor, the system performs the following steps:

1. Spawns a task naming the requested bound unit (the command name). Task spawning implies task creation, which consists of the allocation and initialization of any control structures and data areas required for task control.

2. Calls the Loader to load the requested bound unit.

3. Places a request for the execution of the bound unit against the created task. 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 command execution or by the pressing of the Break key.

4. If the command is Enter Group Request, places a group request against an application task group. The Enter Group Request command terminates. The request is queued if there are other outstanding requests against the application task group from previous Enter Group Request commands.

5. Deletes the spawned task when the command terminates. Optionally issues a ready message to indicate a return to command level.

Command Format

A command line is a string of up to 252 ASCII 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. Arguments, designated by arg, are described below.

A command line can span one or more physical lines. A command 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 canceled by entering a single ampersand on the next physical line.

More than one command can be included in a command line by ending each command (except the last) with a semicolon (;). If any command in the command line is prematurely terminated (interrupt or error), the remaining commands are not processed. Note that only one operator command can appear on a line.

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 (for example, -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 positional arguments. The keyword -ARG must be the last argument of the command line. The arguments following -ARG are passed to the activated (application) task.

PARAMETERS

Arguments are the user-selected items of data passed to a task. In the activated task, which is written in a generalized manner to handle any set of data passed to it, these data items are known as parameters. If the activated task expects positional parameters, the command line arguments passed to it must be in the same order as the task's positional parameters.

SPACES IN COMMAND LINES

Arguments in command lines are separated from each other by spaces. Unless otherwise indicated, a space in command line syntax represents one or more space characters, one or more horizontal tab characters, or a combination of space and horizontal tab characters. You can embed spaces within an argument by enclosing the argument in apostrophe (') or quotation mark (") characters. Note that a file name supplied in an argument can be shown to have a trailing space if the argument is bounded by quotation marks.

PROTECTED STRINGS

Special significance is attached to the following reserved characters:

- Space (blank)
- Horizontal tab
- Quotation mark (")
- Apostrophe (')
- Semicolon (;)
- Ampersand (&)
- Vertical bar (|)
- Left bracket and right bracket ([])

It is occasionally necessary to use a reserved character without its special meaning (for example, a blank could be used in a command argument). The protected string designators (apostrophe and quotation mark) are reserved for this purpose. Reserved characters within a protected string (one surrounded by protected string designators) are treated as ordinary characters. For example, in the argument:

```
-ARG "ALPHA 2" ALPHA
```

the space in ALPHA 2 is treated as part of the name.

Unless the Uppercase command specified that uppercase was off, lowercase characters that are not protected by quotation marks are interpreted as upper case characters. For example, if uppercase is off, the string abcd will be passed to a command as ABCD and the string "abcd" will be passed as abcd.

Use of the protected string designators may also be required when the & character is followed by a number in a command-in file. If &1 does not represent a substitutable parameter, it must be written as &'1' or &"1" (not "&1"). Substitutable parameters are discussed in the Commands manual and in the System User's Guide.

Also, since the protected string designators themselves are reserved characters, it may be desirable at times to suppress their special meaning. To do this, you must enclose the string containing the reserved character in quotation marks or apostrophes. If the reserved character to be protected is the same as the characters enclosing the string, it must be entered twice. For example, to pass the string A"B to a command, you can enter either of the following:

```
"A""B"
'A"B'
```

To pass the string A'B to a command, you can enter one of the following:

```
'A''B'
"A'B"
```

ACTIVE STRINGS AND ACTIVE FUNCTIONS

An active string is a part of a command and is evaluated (executed) immediately by the Command Processor. The resulting value is then substituted for the active string characters in the command line. Any command can be used within an active string. Commands explicitly designed to be used within active strings are called active functions. Active strings and active functions are delimited by left and right brackets.

An example of an active function is [EQUAL a b], which returns TRUE if a is equal to b and FALSE if a is not equal to b. Another example is [LHD], which returns the full pathname of your home directory. If you issued the command:

```
MENU_PR [LHD]>MENUCAT.EN -LC
```

the system would list all the menus in the menu catalog in your home directory.

Active functions can have arguments of their own, and active strings can be nested. For example, the TIME active function (which returns the current time as hour and minute) can be nested in the SUBSTR function (which returns a substring of characters beginning at a specified position and including a specified number of contiguous characters). If TIME returns 10:15, the active string:

```
[SUBSTR [TIME] 4 2]
```

returns the substring 15 (begin at fourth character and return a substring of two characters).

An active string can consist of any number of valid active functions separated by semicolons. The value of the active string is the concatenation of the values of the active functions. For example, if the active string [act_fncl x] returns the value TURN, and the active string [act_fnc2 yz] returns the value OUT, the active string:

```
[act_funl x; act_fnc2 yz]
```

returns the value TURNOUT.

Active strings and active functions are described in detail in the Commands manual.

COMMAND ABBREVIATIONS

The abbreviation processor provides a way of relating a short, user-defined character string to another string of arbitrary length. Suppose, for example, that a program existed in a directory other than your working directory and had several entry points (called ALPHA, BETA, and OMEGA) that performed different functions. Entering the full pathname of the program and its entry point would require you to type an involved command line such as:

```
>PROJECTA>SMITH>SUB_DIR>WIDGET?ALPHA
```

each time you wished to invoke function ALPHA in program WIDGET in directory >PROJECTA>SMITH>SUB_DIR. The abbreviation processor allows you to define a simple name such as ALPHA and equate it to the full command pathname. Similar abbreviations can be defined for functions BETA and OMEGA.

You can create and maintain your own standard sequential file of abbreviations for commonly used commands, control arguments, and pathnames. When the abbreviation processor has been activated by the Abbreviation command, it intercepts a command line, uses your abbreviation file to expand any abbreviations in the command line to their predefined character strings, and then passes the full command line to the command processor.

COMMAND ACCOUNTING

Command accounting is an optional facility that logs all user commands entered through the command language, menus, and the Process Command Line system service macrocall. Commands entered from system groups (those whose first group id character is $) are not logged.

The system records the command's elapsed time and resource usage as well as the group id and user id of the issuer. Refer to the System Building and Administration manual for information on requesting command accounting and obtaining command accounting reports.

COMMAND BEAMING

Command beaming allows you to execute commands in another computer node. (A task group capable of processing the commands you enter must exist in the remote node.) When you issue a Beam command, the system's remote file access facility (described in Section 2) reads your command-in and user-in files and sends the data to the node specified in the Beam command. The output generated at this node is written to your user-out and error-out files. All subsequent commands you issue will be executed at the specified node until you issue another Beam command to return to your node.

All memory space, processor time, and disk space required to execute the commands are distributed to the remote node.

Since command beaming allows you into another computer, you can enter commands to find out the status of users, applications, devices, and so forth on that node. You can queue requests, send messages to local users, and update the node's remote file catalog.

EC AND START_UP.EC FILES

The Command Processor is able to read commands from a source other than an interactive user terminal. One example is an Execute Command (EC) file that you construct through an editor. An EC file is a text file that contains command lines (for input to the Command Processor) and/or Execute command directives. An EC file is read by the Command Processor when:

- The Command Processor is invoked by an Execute command.
- A task group is activated with the Command Processor as its lead task and the EC file is specified as the task group's user-in file.

When you enter a request to have a task run in absentee mode, you specify an EC file that is to be read by the Command Processor (refer to the System User's Guide for further details).

EC Files

An EC file might contain a series of commands that you execute on a frequent basis, such as commands to execute a set of application programs that run at the end of the month to summarize inventory, sales, and accounts receivable. EC files can range from simple to complex. An example of a simple EC file is:

```
ED -PT
FORTRANA AREA -LE
LINKER AREA -IN LNKDR
DPRINT AREA.M
AREA
```

This EC file is made up of commands that are most often used in developing a FORTRAN program called AREA. The ED command invokes the line editor, FORTRANA invokes the the FORTRAN compiler, LINKER invokes the Linker, DPRINT prints the link map, and AREA executes the program.

A more complex EC file uses active functions and substitutable parameters. The following file could be used to create, compile, and link any program. (The lines beginning with the & character are Execute command directives.)

```
& CREATE, COMPILE, AND LINK A &1 PROGRAM
&P BEGIN EDITOR SESSION
ED -PT
&1 &2
&P COMPILATION BEGINS
&IF [EQUAL [RETCODE] 0000] &THEN $ELSE &G ERROR1
&P LINKER SESSION BEGINS
&A
LINKER &2
LINK &2
QT
&IF [EQUAL [RETCODE] 0000] &THEN $ELSE &G ERROR1
&P LINK COMPLETE
&G FINISH
&L ERROR1
&P ERROR ENCOUNTERED IN DEVELOPMENT SEQUENCE
&P EC TERMINATED
&D
&Q
&L FINISH
&D
&Q
```

You could execute this EC file for a COBOL program development session by entering:

```
EC PROG_DEV COBOLA PAYROLL
```

The pathname PROG_DEV is substituted for all occurrences of &0 (none in this example), COBOLA is substituted for all occurrences of &1, and PAYROLL is substituted for all occurrences of &2.

EC files are discussed in detail in the System User's Guide and the Application Developer's Guide.

START_UP.EC Files

A special application of EC files is their use at system initialization and at task group activation.

SYSTEM START_UP.EC FILE

After configuration (after the CLM_USER file of configuration directives is executed), the system searches for a user-written command file named START_UP.EC in the system root directory. If this START_UP.EC file is present, it is executed. A typical system START_UP.EC file might contain operator commands used to establish an application environment for the installation. An example of such a START_UP.EC file is:

```
CBP BUFF1 -NBF 10 -CISZ 1024
CBP BUFF2 -NBF 5 -CISZ 512
CBP BUFF3 -NBF 5 -CISZ 256
CBP BUFF4 -NBF 20 -DIR
START_MAIL
&Q
```

This START_UP.EC file creates several buffer pools of various common sizes and activates the local mail/message facility.

USER START_UP.EC FILE

When a task group whose lead task is the Command Processor is activated, the Command Processor searches for an EC file named working_directory>START_UP.EC. If such a file is present, the Command Processor executes it before performing any other action. This file could contain commands to direct the execution of the tasks of the job and/or perform certain housekeeping tasks. An example of a user START_UP.EC file is:

```
AMM -OFF
ST 2 -EFN PROGA
ST 3 -EFN PROGB
ST 4 -EFN PROGC
```

This START_UP.EC file causes three tasks to be activated and specifies that the user does not want to receive local messages.

Section 4
EXECUTION ENVIRONMENT

System control of user applications and system functions is accomplished within the framework of the task group. A task group consists of a set of related tasks. The most simple case of a task is the execution of code produced by one compilation or assembly of a source program (after the code is linked and loaded).

TASK GROUPS AND TASKS

MOD 400 allows you to configure a system dedicated to interactive applications or to a combination of interactive and absentee 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, you are utilizing multiprogramming. You can step through an application in sequence by causing tasks in the group to be executed one at a time, or you 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, the system builder must define an environment for each application so that the application knows the limits of its resources. This defined environment is called a task group, and its domain includes one or more tasks, a memory pool, files, peripheral devices, and priority levels. By defining the total system environment to consist of more than one task group, the system builder divides up the resources so that more than one application can run concurrently. To do this the system builder divides the memory not occupied by the Executive into one or more user memory pools. Users assign task groups to memory pools at group creation time. Task code of a task group is loaded only into that task group's pool; the task obtains dynamic memory from that pool.

By using the resources of one task group repetitively, you can run an application as a sequence of job or program steps. To do this, invoke the Spawn Group command to create a task group that uses the Command Processor (whose function is to process system-level commands). You can enter commands through task groups whose lead task is the Menu Processor or the Command Processor.

One method of sequencing application steps is to have this spawned group issue a Spawn Task command for each task to be executed. This command causes a 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 group can be executed in sequence. For example:

```
ST 1 -EFN REP_DATA -WAIT      (Spawn task to gather
                               report data and wait for it
                               to complete)

ST 1 -EFN PR_RPT -WAIT        (Spawn task to print report
                               and wait for it to complete)
```

A variation of this procedure can be used to attain multitasking within one task group. Consider the situation in which the Command Processor is the lead task and reads a file containing Spawn Task commands. The Command Processor does not wait for the execution of the individual tasks; rather it continues to spawn tasks until it reads an end-of-file or &Q directive. The spawned tasks are loaded and run concurrently in this task group, contending among themselves for the resources belonging to the group. For example:

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

This method can be used only if a synchronization mechanism such as a semaphore is employed to ensure that PR_RPT does not run until REP_DATA has finished (refer to "Semaphores" in Section 5 for further information). In a multiprocessor system it is possible that the PR_RPT program will be started before REP_DATA has finished gathering its data. In any system, each task is given a certain amount of time to execute, after which it must wait for some event. If the task exceeds this amount of time, the system schedules it to resume after other tasks. It is possible that REP_DATA could be stopped before it has finished collecting the data and that PR_RPT could be started. For reasons such as these, a synchronization mechanism is a necessity.

The Command Processor must be the lead task of an absentee task group so that it can read the EC file containing the desired commands. However, the Command Processor does not have to be the lead task of an interactive task group. An application consisting of one task could execute in a task group whose lead task is the application task. If the application requires step control or multitasking and you do not need to use commands for control, you can generate a task group whose lead task contains the Assembly language system service macrocalls whose functions are analogous to the Create Group, Create Task, Spawn Group, and Spawn Task commands.

These situations are illustrative and do not exhaust the various ways in which you can control program execution.

To summarize, a task group is both the owner of system resources and the context in which system control of tasking is accomplished. A task can 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 Executive, and its execution must be requested through the Executive software.

The source language from which task code is derived can be any of the languages supported by the Executive. Source code is compiled (or assembled) and linked to form bound units consisting of a root and zero or more overlays. (Refer to "Bound Units" later in this section for more information.)

Application Design Benefits of Task Group Use

Designing an application around a task group provides intertask communication and Executive control of multiple unrelated task groups.

INTERTASK COMMUNICATION

The tasks in a task group execute asynchronously under control of the Executive. Tasks within a group can communicate through control structures supplied with each task request for inter-task communication.

Asynchronous tasks provide effective software response to information received from real-time external sources, such as communications or process control systems. Usually, the task (a line protocol handler) 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 Executive to request the execution of the processing task, supplying along with the request a control structure containing a pointer to the new information to be processed. The Executive responds to the request by activating the requested task or by queuing the request if other requests for the execution of the task are still pending.

Communications applications can use a high priority task to respond to data interrupts and determine which processing task should handle the data. This higher priority task uses the 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 higher priority task to scan and update temperature, thickness, or raw material level sensors that monitor the physical status of the process. This information is passed to a processing task with a lower priority that determines the necessary adjustments based on the new data. 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.

SYSTEM CONTROL OF TASK GROUPS

System control of an application based on the use of multiple task groups is important for several reasons. First, these applications can be thought of as consisting of multiple unrelated "jobs" (task groups) made up of one or more "job steps" (tasks). The sequence of task execution can be controlled by the system (Command Processor) as it processes synchronously supplied commands instead of responding only to externally supplied interrupts. The next "step" is started only when the previous step terminates. (You must ensure that the steps will be carried out in order.)

If any one set of tasks does not fully use the available processing time, the system can make more efficient use of resources by rotating their use on the basis of interrupts and priority level assignments.

The use of independent task groups that are subject to system control prevents one task group from adversely affecting another. If an error occurs in one task group, this group can be aborted while the others continue to execute.

To summarize, system control of multiple task groups provides the following advantages:

- Job and step execution sequencing
- Efficient system resource use
- Job independence.

Generating Task Groups and Tasks

The system provides tasking facilities regardless of the source code in which the application is written. Once generated, all tasks are subject to the same system controls, whether written in COBOL, FORTRAN, BASIC, Pascal, C, Ada, or Assembly language. Some languages (such as COBOL and BASIC) do not provide for tasking as part of the programming language's capabilities. In these cases, the generation of tasks consisting of code written in those languages is done through commands. Although tasks written in languages such as Assembly language and FORTRAN can be generated at the control language level, these languages have a facility for generating task groups and tasks without recourse to commands. Assembly language programs use system service macrocalls; FORTRAN programs use tasking routines.

From the overall system viewpoint, the actions of the control language in the generation of task groups and tasks are much more visible than the same capabilities in Assembly language and will be considered next.

As shown in Table 4-1, commands submitted by the operator and commands submitted by other users share some of the task group generation functions and also perform unique functions. The control commands are divided into three groups:

1. Commands that perform the same function whether submitted by the operator or another user. *
2. Commands that can be entered only by the operator.
3. Commands contained within the content of an existing task group request.

Table 4-1. Task Group and Task Functions Possible from Interactive and Absentee Modes

Function	User Commands		Operator Commands	
	Interactive	Absentee	Interactive	Absentee
Create Group	Yes	No	Yes	Yes
Enter Group Request	Yes	Yes	Yes	Yes
Delete Group	Yes	No	Yes	Yes
Abort Group	Yes	No	Yes	Yes
Spawn Group	Yes	No	Yes	Yes
Bye	Yes	Yes	No	No
Suspend Group	Only operator commands exist for these functions.		Yes	Yes
Activate Group			Yes	Yes
Abort Group Request			Yes	Yes
Create Group Request Queue			Yes	Yes
Create Task	Yes	Yes	Only user commands exist for these functions.	
Delete Task	Yes	Yes		
Enter Task Request	Yes	Yes		
Spawn Task	Yes	Yes		

NOTE

The Command Processor executes in both interactive and/or absentee mode.

Characteristics of Task Groups and Tasks

Task groups and individual tasks can be originated in either of two ways: by creation or by spawning. The choice depends on application design considerations as well as the intended functions.

There are important differences between tasks (and task groups) that are generated by a create function and those originated by a spawn function. Created task groups and tasks are permanent; they remain available in memory until explicitly removed. Spawned task groups and tasks are transitory; they perform a function and disappear.

Created task groups and tasks are passive; they must be explicitly requested to execute in order to perform their intended function. Spawned task groups and tasks cannot be requested. The spawning of a task group or task is equivalent to a create-request-delete sequence of control language commands. In a spawn operation, the task group or task is defined, provided with system resources and control structures, executes, terminates, and has its resources deallocated, all in one continuous process.

FORTRAN or Assembly task code may cause extensive action in its own behalf, as when application task code requests a system service or the execution of another task while awaiting the completion of the requested task. Each task that requests another supplies the address of a control structure through which the issuing task and the requested task can communicate, and which the Executive uses to coordinate task processing.

Task Group Identification

Each task group has a unique identifier. Honeywell-supplied system task group identifiers begin with a $ as shown in Table 4-2 below:

Table 4-2. System Task Group Identifiers

Task Group ID	Function
$D	Debug
$L	Listener
$P	Deferred Print
$S	System
NOTE The Multiuser Debugger does not require the dedicated system task group $D.	

The identifier for a user task group in the Create Group or Spawn Group command is a 2-character name that should not have the dollar sign ($) as its first character. The identifier (or group-id) can be indicated or implied in commands to designate what task group is to be acted upon. The operator can include the task group identifier when responding to messages from the task group.

MEMORY MANAGEMENT AND PROTECTION

The system (hardware and software) provides a memory management and protection facility that performs the following functions:

- Allocates memory to guarantee each task group (user) its own address space.
- Protects multiple users from each other and the system from the users.

The hardware used to provide memory management and protection is called a memory management unit. The type of memory management unit varies according to the kind of processor. DPS 6 systems use either the Basic Memory Management Unit (BMMU) or the Extended Memory Management Unit (EMMU). Each of these memory management units is based on the concept of segmentation.

Segmentation

The memory management unit maps a segmented address space onto physical memory. The unit of memory allocation is a segment. A segment is a variably sized area of memory that usually consists of a logical entity such as a procedure. The system memory management and protection facility treats all addresses generated by the central processor as segment-relative addresses. It maps the segment-relative addresses through the memory management unit to absolute physical addresses. No segment can be less than 512 bytes in length. Segment size is always a multiple of 512 bytes.

SEGMENTATION WITH BASIC MEMORY MANAGEMENT UNIT

The BMMU supports up to 31 segments, 16 of which can be up to 8K bytes (K=1024) in size and 15 of which can be up to 128K bytes in size. The segments that can be up to 8K bytes are called "small segments"; those that can be up 128K bytes are called "large segments." The 16 small segments are numbered from 0.0 through 0.F; the 15 large segments are numbered from 1 through F. All small segments, and often some large segments, are reserved for system use; the actual number reserved is established at system generation.

The BMMU provides a total of 2 million bytes of segmented address space for each task.

Each segment is described by a 4-byte segment descriptor that contains the segment's starting physical address, its length (in units of 512 bytes), and its access rights for each ring (refer to "Segment Ring Protection" below).

Although you can assign any of the large segments to a bound unit when it is linked, the availability of a segment depends on the system configuration. Therefore, most applications simply let the system assign segment numbers. The identity of the segments available to you should be obtained from the system administrator.

SEGMENTATION WITH EXTENDED MEMORY MANAGEMENT UNIT

The EMMU supports up to 256 segments, each of which can be up to 128K bytes in size. The segments are numbered from 00 through FF. Segments 00 through 7F are reserved for system use; segments 80 through FF are available for user tasks.

The EMMU provides a total of 32 million bytes of segmented address space for each task.

Each segment is described by a 2-byte segment descriptor that contains the segment's starting physical address, its length (in units of 512 bytes), and its access rights for each ring (refer to "Segment Ring Protection" below).

Segment Ring Protection

Access to memory segments is controlled through the memory management unit. The memory management unit assigns each executing task to a ring of privilege. (Rings may be thought of as concentric circles, like a target. The innermost circle, ring 0, has the most privilege.) During the linking of a bound unit, you can assign access attributes to each bound unit to indicate whether a task executing in a particular ring of privilege can read, write, and/or execute in the code or data segment of the bound unit. However, it is recommended that you use the system defaults.

System tasks execute in ring 0 (privileged state). User tasks can execute in rings 1, 2, and 3. Ring 0 is most privileged, and ring 3 is least privileged. The ring in which a user task executes is defined by the type of memory pool to which the task has been assigned (refer to "Ring Access Rights" later in this section).

Every attempted access to a segment is checked for access rights in the executing task's ring of privilege. The system compares the ring number of the executing task with the access attributes of the segment to be accessed. An access violation trap occurs if a user application attempts to access one of its segments without having the proper segment access rights.

MEMORY POOLS

At system startup the Configuration Load Manager (CLM) reads a file of directives, sets up memory pools from the supplied specifications, and indicates to the Loader what system and user-written software is to be resident for the life of the system. On the DPS 6/22, the Autoconfigurator creates the file. On other systems, the system builder can use the line editor to create the file.

After the system has been in operation for a while, experience may show the desirability of reconfiguring the pool sizes so that they meet user requirements more precisely. The system builder can hand-tailor the MEMPOOL and SWAPPOOL directives in the CLM file using the line editor. Refer to the System Building and Administration manual for information on the CLM directives.

MOD 400 supports the following types of memory pools:

- System pool - Contains the system task group and all globally shared elements. There is only one system pool per system.

- Swap pool - Provides an environment in which segments can be swapped out to disk to make room for competing users, and in which the memory requirements of individual users do not have to be predetermined. Systems with EMMUs should have all of user memory as one swap pool. Systems with BMMUs should have all of user memory as multiple swap pools.

- Independent pool - Provides an environment suitable for applications with well-defined memory requirements, all of whose tasks must must be in memory at the same time. There can be multiple independent pools in the system.

Swap and independent pools allow applications running on systems with a BMMU to access more than 2 million bytes of physical memory.

If multiple swap or independent pools are configured, Listener can optionally ensure an even distribution of task groups among memory pools by assigning each new user to the memory pool with the fewest task groups.

Sharing Memory Pools

Swap and independent pools will be shared if users assign more than one task group to the same pool. As the tasks execute, they contend for the same memory space. Tasks running in an independent pool should be designed so that they can be suspended or take some alternative action when no additional memory is available.

Memory Pool Attributes

Memory pools can have one or more of the following attributes: protection, containment, privilege, serial-usage, and ring access rights.

PROTECTION

When a memory pool has the protection attribute, it cannot be written into by a task running in another pool. The Executive uses the memory management unit to prevent all write intrusions by foreign tasks. A task attempting to write into a protected pool receives an error notification from the Executive.

Protection applies to memory pools and not to task groups. Groups sharing a a memory pool are protected from each other only in the swap pool. Tasks within a group are not protected from each other.

All pools are automatically generated as protected; this attribute cannot be changed.

CONTAINMENT

When a memory pool has the containment attribute, tasks running in the pool cannot write outside the pool area. The Executive uses the memory management unit to prevent all tasks from writing outside the pool. A task attempting to write outside of a contained pool receives an error notification from the Executive.

The system pool cannot be contained. Swap and independent pools are automatically generated as contained; this attribute cannot be changed.

PRIVILEGE

When a memory pool has the privilege attribute, any task running in that pool can execute privileged instructions. The following Assembly language instructions are privileged:

ASD	IO	LEV	WDTF
CNFG	IOH	RTCF	WDTN
HLT	IOLD	RTCN	

If the pool does not have the privilege attribute, any task attempting to execute one of the above instructions will trap.

The system pool is always privileged; this attribute cannot be changed. Swap and independent pools are unprivileged; this attribute can be changed in the CLM MEMPOOL directive.

SERIAL USAGE

When a memory pool has the serial-usage attribute, it can be used by only one task group at a time.

The system and swap pools are generated without the serial-usage attribute; this specification cannot be changed. Independent pools can be specified in the CLM MEMPOOL directive as having the serial usage attribute.

RING ACCESS RIGHTS

Each type of memory pool is automatically assigned a ring access designation. There are four rings, numbered 0 through 3. Rings 0 and 1 are privileged rings; rings 2 and 3 are unprivileged. Tasks acquire the ring attribute of the pool to which their task group is assigned. The pools and their associated rings are:

Pool	Ring
System	0
Swap	3
Independent	1 or 2

Independent pools have ring 1 access if they are privileged and ring 2 access if they are not privileged.

Ring access is used with segment ring protection to determine the ability of a task to access memory (refer to "Segment Ring Protection" earlier in this section). A task whose ring access is 3 can only access memory protected at ring 3. A task whose ring access is 2 can only access memory protected at rings 2 and 3. A task whose ring access is 1 can access memory protected at rings 1, 2, and 3. A task whose ring access is 0 can access memory protected at rings 0, 1, 2, and 3.

The following paragraphs describe each type of memory pool in greater detail.

System Pool

The system pool contains the system task group ($S), certain file control structures, and system elements that are to be shared. Its maximum size is 4 million bytes. The system pool is protected, privileged, and not contained. Tasks running in this pool have ring 0 access.

User task groups cannot be created in the system pool. User tasks cannot execute in the system pool with ring 0 access.

The system task group cannot be aborted or suspended. Since it never terminates, it cannot be requested. The system group always has read and write access to all of memory. It handles all system dialog (including operator commands) through the CLM-designated operator terminal.

The file control structures in the system pool are the File Description Blocks (FDBs) and the buffers for sharable files. Other system elements in this pool include:

- Current function invoked by an operator command.
- Extended Trap Save Areas (TSAs) needed during processing.
- Control blocks for all tasks (TCBs) and task groups (GCBs).
- Globally sharable bound units.
- File System directory and file definition blocks.
- Public buffer pools.
- Memory control blocks for swap pool segments.

Swap Pools

A swap pool is a privileged, protected, and contained pool in which segments can be swapped out to disk in order to make physical memory available to competing users. Swap pool memory management can move segments to physical memory in order to eliminate fragmentation and consolidate available memory space.

Swap pools support both interactive and absentee processing.

The size of a swap pool, plus the size of the Executive and its structures, cannot exceed 2 million bytes in a system having a BMMU and 16 million bytes in a system having an EMMU. On systems with a BMMU, the system builder can configure multiple swap pools. On systems with an EMMU, all of user memory should be one swap pool.

A swap pool is a separate virtual view of the system. That is, each task in a swap pool can view that portion of the pool relating to itself and can view all of the global system space. Tasks running in the swap pool have ring 3 access.

The system acquires space for a given segment from whatever portion of free swap pool memory is available. If not enough space is available for the needed segment, the system attempts to obtain memory by swapping out lower priority tasks in the same pool that are waiting on an event. If this action does not produce enough memory, the requesting task is swapped out until sufficient space becomes available. A task is swapped out under one of the following conditions:

- If it is waiting on an event that is of potentially long duration and swap pool memory is required by a competing task.
- If memory is required to roll in a higher priority task.
- If the task has been suspended by the operator.

A task is swapped back in when the swap pool memory is available. The task may be swapped in immediately, it may be swapped in after tasks waiting on events of long duration are swapped out, or it may be swapped in after lower priority tasks are swapped out. A task is swapped back in when any event on which it was waiting has completed or when it is reactivated by an operator.

The entire context of a task in the task group must be in memory for the task to execute. Other tasks in the task group need not be in memory. However, thrashing may occur if too many users are assigned to too small a swap pool.

A task in a swap pool can overwrite shared data designated for writing. A task cannot overwrite another task, another task's data, or sharable read-only data. Further, tasks in a swap pool can only read (not write) system structures.

Tasks running in a swap pool have logical elements (for example, bound units) equated with segments. The Executive aligns the logical elements on segment boundaries. This configuration is represented in Figure 4-1.

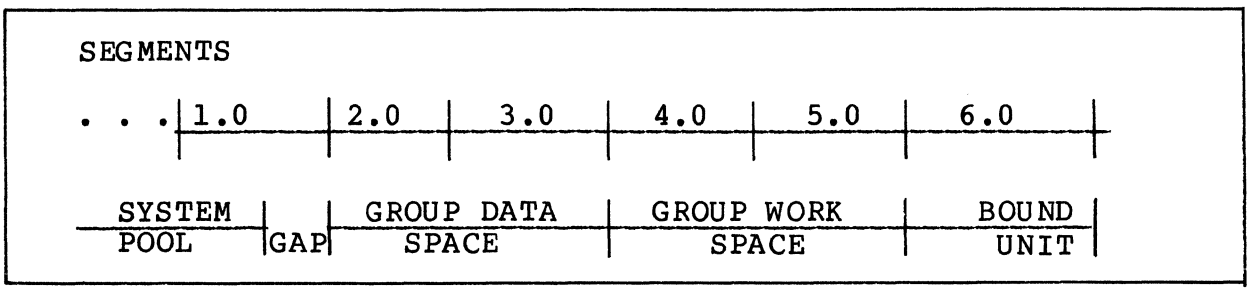


Figure 4-1. Sample Swap Pool Group Segment Assignments

Each task group in the swap pool has group global space that cannot be accessed by any other group. Each task in a swap pool group can have task private space that cannot be accessed by any other task. For further information refer to "Swap Pool Task Address Space" later in this section.

Independent Pools

An independent pool is protected and contained; it can be privileged or not and serial-usage or not. Tasks running in an independent pool have ring 2 access if the pool is unprivileged and ring 1 access if it is privileged.

Independent pools support both interactive and absentee processing.

The size of an independent pool, plus the size of the Executive and its structures, cannot exceed 2 million bytes in a system having a BMMU and 16 million bytes in a system having an EMMU. On larger systems, multiple independent pools can be configured.

It is important that the memory requirements of the group(s) using an independent pool be estimated carefully because the entire context of a task group must be in memory for a task in the group to execute. It is possible that task group memory requirements may exceed the size of the memory pool because of memory fragmentation.

Note that even with protection and containment, a task in an independent memory pool can accidentally overwrite code or data belonging to its own or another group in the pool.

Each independent pool is a separate virtual view of the system. That is, each task in an independent pool can view that portion of the pool relating to itself and can view all of the global system space. Thus, a task in an independent pool can reference a memory location in that pool and in system global space.

Independent pools are designed for applications that you may not want to execute in a swap pool.

The system aligns user memory pools on segment boundaries (multiples of 512 bytes). This configuration is shown in Figure 4-2.

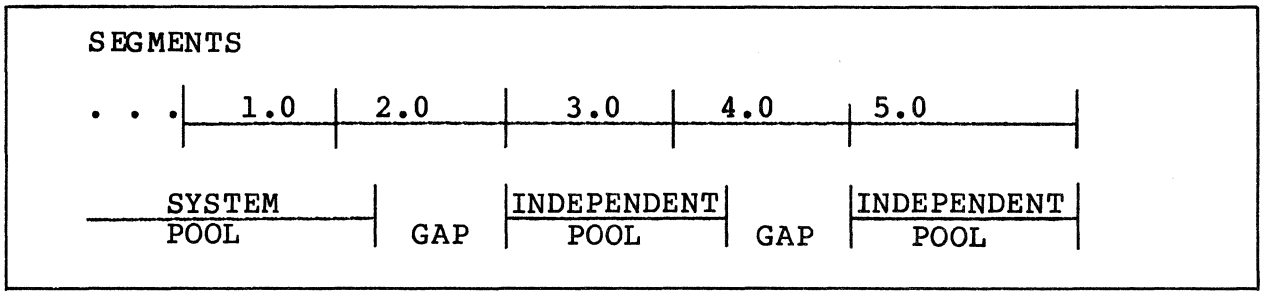


Figure 4-2. Sample Independent Pool Group Segment Assignments

Selecting Memory Pool Types

The different types of memory pools provide you with the means to respond to the unique demands of multiple application programs. Through the use of memory pools, you can exercise control over memory usage and, at the same time, provide individual task groups with specialized protection.

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. The following points should be kept in mind:

- All systems must have a system pool.
- In systems with a BMMU, all user memory should be devoted to multiple swap pools.
- In systems with an EMMU, all user memory should be one large swap pool.
- One or more independent pools can be selected for applications that you do not want to run in a swap pool.

If you do not configure any memory pools, you will be provided with one swap pool whose size is all of memory, less the amount of memory occupied by the system pool.

Memory Pool Layout

To obtain efficient use of memory and of the memory management unit, the CLM sorts the memory pools specified in a configuration as follows:

1. The system pool is configured in the first available memory after the system data structures. The system pool cannot exceed 4 million bytes.
2. If swap pools are configured, they follow the system pool.
3. Independent pools are configured after all swap pools.

Fixed System Area

After the configuration process is complete, the following software components and data structures are located in the fixed system area of memory:

- Basic Executive plus resident overlays
- User-written or vendor-supplied extensions to the Executive
- Device drivers
- Intermediate request blocks needed for task groups
- Trap save areas
- Overlay area(s) for system software
- File control structures.

The fixed system area is static. Unlike the other memory areas whose contents can vary dynamically, its structure remains the same for the life of the system. 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.

BOUND UNIT CHARACTERISTICS

Task code is derived from programs written in a source language and compiled or assembled to form object units (also called compilation 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.

General Bound Unit Characteristics

Bound units have the following general characteristics:

- Each bound unit consists of a root segment and any related overlays.
- A load element is composed of one or more object units.
- The initial load element is called the root; it must be resident when the bound unit is being executed.
- A load element that replaces another load element when loaded into memory is called an overlay.

You can direct the Linker to perform the following actions on bound units:

- Map the code and data into the same load element or into separate elements. If the bound unit is to be reentrant, the code and data must be in separate load elements.

- Specify ring access rights. If the bound unit is to be reentrant, the default access attributes are ring 3 read and execute access for both code and data, ring 0 write access for the code segment, and ring 3 write access for the data segment.

- Associate a specific segment number or numbers with a bound unit. Normally, the Linker assigns default segment numbers. If you assign segment numbers at link time, you must be careful to avoid segment conflicts in the configuration and application environment in which the bound unit is to run so as to avoid inefficient loading.

Your physical address space is not necessarily contiguous. Memory requirements are satisfied on a segment basis rather than on a user basis.

Note that for systems with a BMMU, you have a maximum of 11 large segments available when constructing a task's address space. Frequently, fewer large segments will be available, depending on the system configuration.

Sharable Bound Units

The use of sharable 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, sharable bound units are transient in memory and are loaded during processing. A usage counter is incremented each time a request is made for the bound unit, and decremented each time a request is completed. The unit remains in memory as long as a task is using the code. As soon as the usage counter is decremented to zero, the space occupied by the bound unit is returned to available status.

SHARABLE BOUND UNITS IN SWAP POOLS

If a bound unit is sharable only within a swap pool, its root segment descriptor is placed in a portion of memory where it is accessible to all tasks in that pool. The bound unit should have no fixed overlays; floatable overlays can be shared if an OAT is used. (See "Bound Unit Overlays" later in this section.) To be recognized as sharable by the Loader, and to be loaded into a user memory area, the bound unit must have been linked using the SHARE directive.

Additionally, task private segments are shared if the task forks. (A task forks if it issues a Create Task or Spawn Task command with an entry point address rather than a pathname definition.) Forked tasks share the same segments; they have the same access to and copy of the forked segments until one task modifies its address space. (Address space defines a task's boundaries in a swap pool. Refer to "Swap Pool Task Address Space" later in this section.)

SHARABLE BOUND UNITS IN INDEPENDENT POOLS

In an independent pool, bound units can be sharable by tasks within a task group or can be sharable by all task groups. Sharability is established when the bound unit is linked. To be sharable by tasks and task groups within a pool, the bound unit must be linked with the SHARE directive. Bound units linked with the SHARE directive are loaded into the requesting task group's memory pool.

GLOBALLY SHARABLE BOUND UNITS

To be sharable by task groups in other pools (globally sharable), the bound unit must be linked with the GSHARE directive. Bound units linked with the GSHARE directive are loaded into the system pool. Since system pool memory is a critical resource, the use of globally sharable bound units requires careful planning and control. If all of user memory is one large swap pool, the SHARE directive has the same effect as the GSHARE directive, and does not clutter up the system pool.

Operator commands can be used to load and unload globally sharable bound units.

SHARABLE BOUND UNITS AND EXECUTIVE EXTENSIONS

Sharable bound units and the Executive extensions that are loaded through LDBU directives when the system is configured differ in one major respect. Executive extensions can be accessed symbolically by any task, but a sharable bound unit must be accessed as a bound unit.

When an Executive extension is loaded during system configuration and is made permanently resident by an LDBU directive, its symbols are included in the system symbol table. Since a sharable bound unit is transient and is loaded after the system has been configured, no entry is made for it in the system symbol table. For this reason, it must be accessed as a bound unit. Table 4-3 compares permanently resident Executive extensions and transient sharable bound units.

Table 4-3. Comparison of Executive Extensions and Sharable Bound Units

Characteristics	Executive Extension	Sharable Bound Units
Multiple users	Yes	Yes
Permanently resident (fixed area)	Yes	No
Temporarily resident (dynamic area)	No	Yes
Symbols in system table	Yes	No
Accessed symbolically	Yes	No
Have overlays	No	Yes
Called by bound unit name	No	Yes

NOTES

1. If the extension is an Assembly language bound unit, it may have within it sections of code or control structures controlled by semaphores that would be accessible to other Assembly language tasks (refer to "Semaphores" in Section 5 for further information).

2. Overlays are not sharable unless Overlay Area Tables (OATs) are used (refer to "Bound Unit Overlays" below).

3. The Executive does not "remember" extensions by their names. A request for an extension by name results in another copy being brought into memory.

Bound Unit Search Rules

The Loader uses search rules to locate a bound unit to be loaded. The Loader starts the search in response to a command containing an argument naming the bound unit to be loaded.

The rules that regulate the search process define three directory pathnames and the sequence in which they are used during a search. The pathname sequence is as follows:

1. User task group working directory.

2. System directory -LIB1 argument of the Change System Directory command.

3. System directory -LIB2 argument of the Change System Directory command.

The Change System Directory command can be used to change pathnames associated with system directory arguments -LIB1 and -LIB2. The pathname of a user's working directory is established through a Change Working Directory command or through the -WD argument of the Enter Group Request or Spawn Group command. For login users, the -WD argument of the Spawn Group command issued by the Listener is taken from the -HD argument in the Login command line.

Bound Unit Overlays

In smaller systems, you may need to minimize the amount of memory required to execute a bound unit containing application code. You can accomplish this by directing the Linker to create the bound unit as a series of overlays (separately loadable pieces) so that the entire bound unit does not have to be resident at one time. Each bound unit consists of a root and, optionally, one or more related overlays. The system loads the bound unit root automatically when the bound unit is invoked. Overlay loading is controlled by the application itself. The maximum number of overlays is 65,536. The use of overlays requires careful planning so that required code is not lost or repetitively loaded.

Two types of overlays are available for your use: nonfloatable and floatable. In addition, you can use overlay areas to control the placement of floatable overlays.

NONFLOATABLE AND FLOATABLE OVERLAYS

Overlays can be loaded at a fixed displacement from the base of the root (nonfloatable overlay) or into a block of memory allocated explicitly by you or implicitly by the system (floatable overlay).

Nonfloatable Overlays

A nonfloatable overlay is loaded into the same memory location relative to the root each time it is requested. Object units whose code is to be loaded as nonfloatable overlays must be defined as fixed overlays by the Linker OVLY directive. When the root of a bound unit having fixed overlays is loaded, the Loader allocates a container (segment or memory block) large enough to hold the root and all of its fixed overlays.

Assembly language programs can use system service macrocalls to load and execute nonfloatable overlays. COBOL programs can use CALL/CANCEL statements to control nonfloatable overlays. FORTRANA and Pascal provide overlay handlers as part of the run-time libraries. BASIC programs must link a user-written Assembly language overlay manager with the application program, since the BASIC language does not supply this functionality.

Floatable Overlays

A floatable overlay is linked without having a fixed relative location to the base of the root. It can be loaded into any available memory location. Floatable overlays must meet the following criteria:

1. The overlay must not contain external definitions referenced by the root or another overlay.

2. The overlay must not make displacement references to the root or any other overlay.

3. The overlay must not contain external displacement references that are not resolved by the Linker.

The application program can use one or more areas of available memory for the placement of floatable overlays. The program can deal with memory management in one of the following ways:

1. Allow the system to place the overlay in an available memory block allocated from the user's independent memory pool or, if loaded in a swap pool, from group work space. (Group work space is a segment common to all tasks in a group. Refer to "Swap Pool Task Address Space" later in this section.)

2. Create a set of overlay areas using system service macrocalls, and allow the system to manage the areas and locate the requested overlays. In an independent memory pool, the overlay areas are created from a memory block in the pool. In a swap pool, the segment(s) allocated for the root are expanded to contain the created overlay area.

3. Perform its own memory management by linking a user-written Assembly language overlay manager with the root of the bound unit. In an independent memory pool, you may choose to have the overlay occupy part or all of a memory block. In a swap pool, you may choose to have the overlay occupy part or all of a segment.

Linking Floatable and Nonfloatable Overlays

Floatable and nonfloatable overlays are defined through the Linker. When using the Linker, forward references can be made to symbols defined in object units to be linked later (the rules for floatable overlays must be observed). Backward references can be made to symbols previously defined, provided 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 and unprotected symbols that are at locations equal to or greater than the location designated in the BASE directive, you must take one of the following actions:

- Define all symbols that are to be preserved in a part of the root that is not overlaid.
- Protect the symbols to be preserved by using the Linker PROTECT directive.

A floatable overlay can refer to fixed addresses in the root, in a nonfloatable overlay, or in itself, but cannot refer to addresses in another floatable overlay.

When a root or 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 by searching the system symbol table (that is, the resident bound unit attribute table). The Loader cannot resolve any references to symbols that do not exist in the symbol table. (Linker symbol tables do not exist at load time.)

Figure 4-3 shows the relative location in memory of memory pool AA. Figure 4-4 is the layout of overlays in memory pool AA. 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 Figure 4-4, this memory area begins at the base 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.

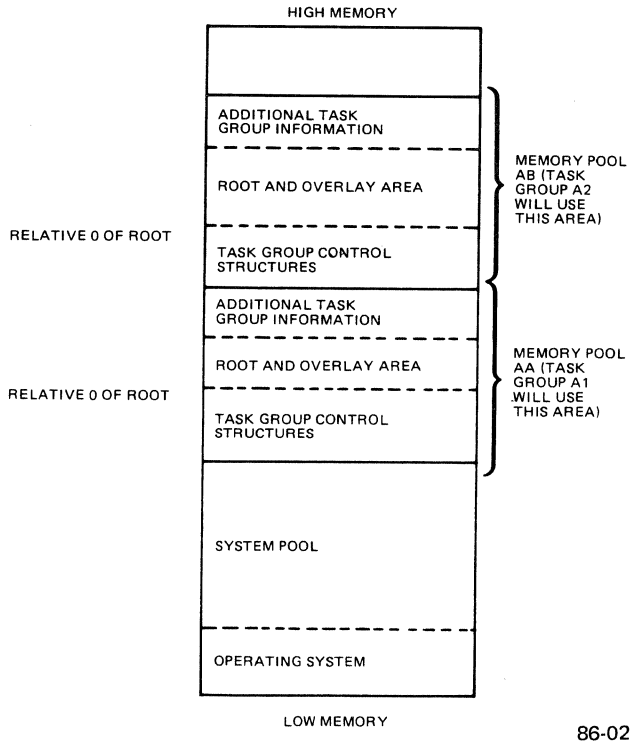


Figure 4-3. Relative Location in Memory of Memory Pool AA

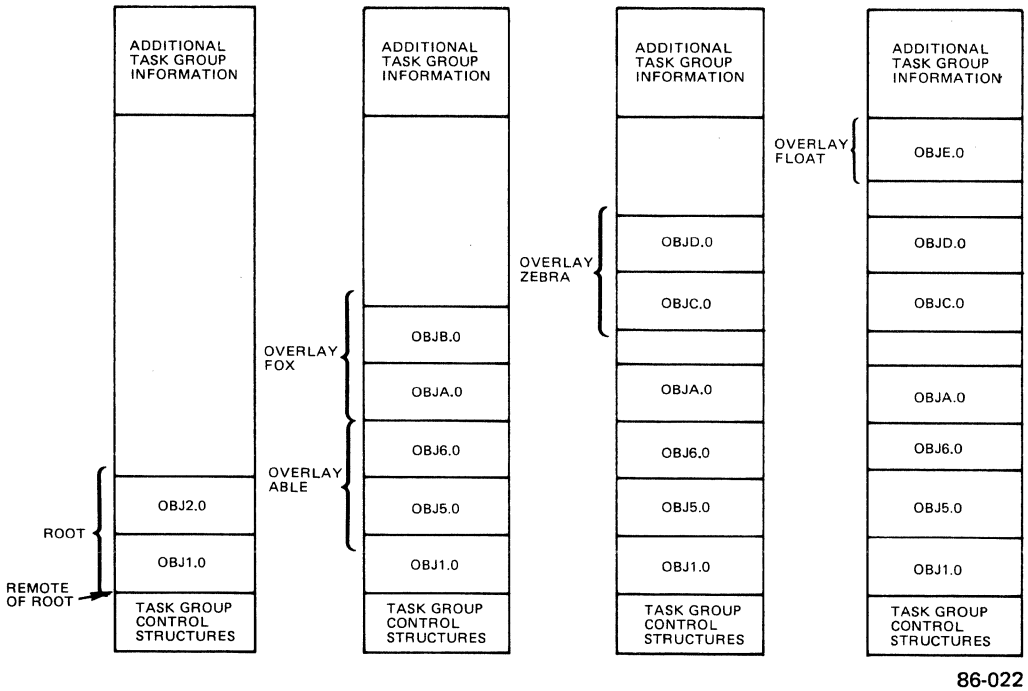


Figure 4-4. Overlays in Memory Pool AA

OVERLAY AREAS

Only floatable overlays can be associated with overlay areas. Overlay areas are a mechanism that allows you to control the placement of floatable overlays without being required to write your own overlay manager.

Overlay areas are fixed size areas of memory whose use is controlled through an Overlay Area Table (OAT). If the bound unit is sharable, the overlays can be shared with other tasks in the task group or with tasks in other task groups. Overlays can also be shared if the bound unit is replicated through the -SHARE argument of the Create Task command. *

You create an OAT through the Create Overlay Area Table system service macrocall. You reserve an overlay area and execute the overlay through an Overlay Reserve and Execute macrocall. You exit from the overlay through an Overlay Release macrocall.

As an example of overlay area use, assume that you desire to share both the root and the overlays of a sharable bound unit whose structure is shown in Figure 4-5.

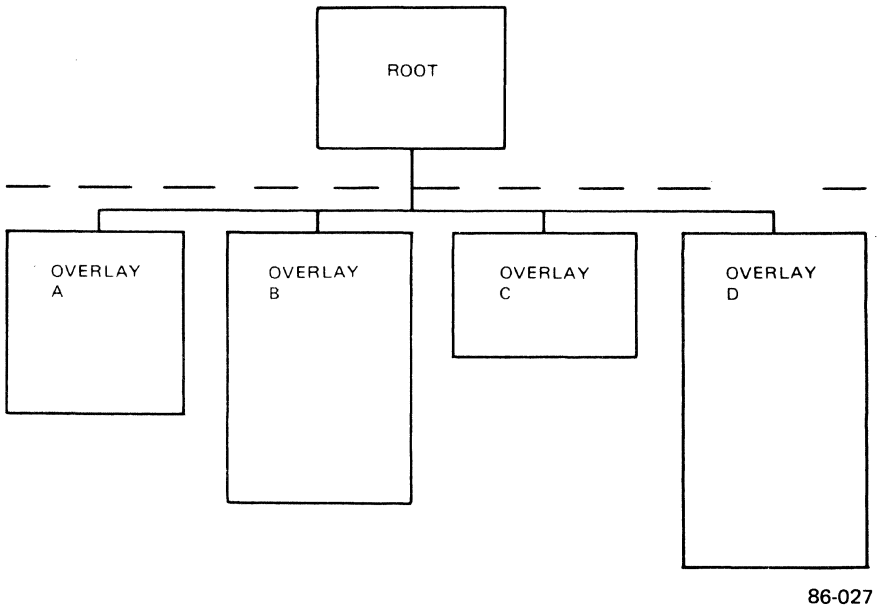


Figure 4-5. Sample Bound Unit Structure for Overlay Area Use

Assume further that tasks 1, 2, and 3 (of the same or another task group) are executing the sharable bound unit and that task 1 has encountered a create OAT function while executing the root.

When the create OAT function is encountered, an overlay area (controlled by the OAT) is created for the task group. In this example, the overlay area has three entries, each entry being 512 bytes long. There is no direct relationship between the number of overlays to be shared and the number of entries in the overlay area. The entries in an overlay area are of equal size. You must create overlay areas large enough to contain the largest overlay (overlay D in this example). The overlay area reserved is depicted below.

ENTRY 1	ENTRY 2	ENTRY 3
512 BYTES	512 BYTES	512 BYTES

When task 2 (or task 3) executes the same create OAT request (that is, when it executes the root), the task is given the address of the OAT already existing in memory.

Assume that task 1 issues an Overlay Reserve macrocall to reserve an overlay area defined by the OAT and to load overlay A in that area. The code and/or data composing overlay A will be loaded in the first free overlay area, and task 1 will be given access to this area. At this instant the status of the overlay area is as follows:

ENTRY 1	ENTRY 2	ENTRY 3
OVERLAY A USAGE = 1	USAGE = 0	USAGE = 0

TASK 1

When tasks 2 and 3 now perform the request for overlay A, they will be given access to the existing copy of the overlay. At this instant, the status of the overlay area is as follows:

ENTRY 1	ENTRY 2	ENTRY 3
OVERLAY A USAGE = 3	USAGE = 0	USAGE = 0

TASKS 1,2,3

Task 2 now requests overlay D. Since a task cannot have more than one overlay in an overlay area at any time, task 2 must explicitly release overlay A before requesting the loading of overlay D. The result of releasing overlay A and requesting overlay D is as follows:

ENTRY 1	ENTRY 2	ENTRY 3
OVERLAY A USAGE = 2	OVERLAY D USAGE = 1	USAGE = 0
TASKS 1,3	TASK 2	

A request by task 3 for overlay C will result in the following situation:

ENTRY 1	ENTRY 2	ENTRY 3
OVERLAY A USAGE = 1	OVERLAY D USAGE = 1	OVERLAY C USAGE = 1
TASK 1	TASK 2	TASK 3

If there were another task in the group (for example, task 4), and the task were to request overlay B, it would have to wait until one of the overlay areas was freed (by an Overlay Release macrocall). If task 4 requested overlay A, C, or D, the task would be given access to the loaded copy of the overlay.

Note that at any given instant several OATs, controlling several different overlay areas, may exist. Even if a task is sharing overlays in different overlay areas, it cannot reference more than one overlay area at any given time. The task must release an overlay in an OAT prior to requesting an area for another.

You use an Overlay Area Release macrocall to exit from an overlay. When this call is executed, the count of the number of users of the overlay is decremented in the defining OAT. When the count drops to zero, the overlay area is marked as available and can be reused by an Overlay Reserve and Execute function.

Bound Unit Allocation

Each task is associated with at least one bound unit. The initial bound unit with which a task is associated is specified at the time the task is created or spawned. At this time, the segment is created/allocated in memory, and the root is loaded in this segment.

If the bound unit was designated as sharable at link time and is currently residing in memory, no loading takes place. The requesting task shares the bound unit already in memory, and the bound unit user count is increased by one. If the bound unit is not in memory, it is loaded.

Execution of a task begins with the specified bound unit. During the execution of this bound unit, the Assembly language user can employ system service macrocalls to load or attach another bound unit. Loading or attaching a bound unit causes the allocation and loading of the segment containing the root of the requested bound unit. (The difference between loading and attaching is that loading returns the entry point of the root segment to the issuing task, while attaching starts the execution of the bound unit root segment at the entry point.) Up to eight bound unit units can be attached. In BMMU systems, the availability of segment descriptors may limit you to fewer than eight attached bound units.

During its execution, a task can issue a system service macrocall to request the creation of a segment to be associated with the task's initial bound unit or any other of its attached/loaded bound units. The macrocall can either specify a segment number or allow the system to select the number in accordance with the specified size.

The allocation of memory for a bound unit depends on whether the bound unit is nonsharable or sharable.

For a nonsharable bound unit, each logical segment is uniquely mapped to a physical segment in memory. Unless the task is forked or the segment is in an OAT, two or more tasks wishing to concurrently use a nonsharable bound unit each receive a copy of the bound unit.

*

If more than one task is executing a pool-sharable bound unit, only one copy of the segment containing the root is allocated in the pool. All tasks use this single copy. Overlays of the bound unit can be shared if an OAT is used. If the bound unit was separated into a code element and a data element, only the code element is shared. Except for forked tasks, each user has a separate copy of the data element.

The Memory Manager assigns a segment number (or numbers) based on the segment descriptors available to the task that initially loads the bound unit. Concurrent users must access the bound unit under the same segment numbers. If the segment utilization of the second and subsequent tasks that attempt to load the sharable bound unit conflicts with its segment number or assignment, an error is returned when the tasks attempt to load the bound unit. In this case, the tasks are not given addressability to the sharable bound unit.

Memory Deallocation

Assembly language users can explicitly deallocate a user-created segment by issuing a Delete Segment macrocall. Users can deallocate bound units by issuing a Detach Bound Unit macrocall for any but the initially assigned bound unit. A segment can be implicitly deallocated from physical memory as the result of the task being deleted or swapped out. It is reallocated when the task is swapped back in.

Overlay areas and defining OATs are deallocated when the last usage of a sharable bound unit has terminated.

SWAP POOL TASK ADDRESS SPACE

Task address space defines a task's boundaries in the swap pool; that is, its visibility within the collection of tasks executing in the pool. The following elements constitute a task's address space:

- Bound unit
- User stack area
- Dynamically created segments
- Group work space
- Group system space
- System global space.

Bound Unit

During its execution life, a task executes one or more bound units. The initial bound unit to be executed is the one specified when the task is created or spawned. In Assembly language programs, other bound units (if any) can be attached or loaded through the Bound Unit Attach or Bound Unit Load macrocalls.

User Stack Area

The user stack area is available to users as a work area through the hardware stack instructions.

Dynamically Created Segments

During execution, a task can extend its address space by creating segments. Assembly language programs use the Create Segment macrocall for this purpose. These dynamically created segments become part of the issuing task's address space.

Group Work Space

The group work space is common to all tasks in a given task group. Assembly language programs can obtain blocks of memory from the group work space when they issue Get Memory macrocalls. All tasks in the task group have read, write, and execute access to the group work space.

The group work space can occupy up to two 128K-byte segments. The group work space grows dynamically, as requests for memory are issued. In both BMMU and EMMU systems, the maximum size is 256K bytes. However, this maximum is reduced to 128K bytes if the adjacent segment descriptor has been allocated.

Group System Space

One group system space is provided for each task group. The system control structures used to support a task group and its member tasks (for example, file control blocks, bound unit descriptors for nonsharable bound units, logical file tables, and logical resource tables) are allocated from the group system space.

The group system space can occupy up to two 128K-byte segments. The group system space grows dynamically, as requests for memory are issued. In both BMMU and EMMU systems, the maximum size is 256K bytes. However, this maximum is reduced to 128K bytes if the adjacent segment descriptor has been allocated.

System Global Space

System global space consists of the fixed system area (permanently configured memory) and the system memory pool. A task's address space includes the segments required for system global space. System code and data are distributed in the task address space.

System Representation of Task Address Space

Figures 4-6 and 4-7 are examples of the mechanism used by the system to represent a task's address space. Figure 4-6 is an example of a system having a BMMU; Figure 4-7 is an example of a system having an EMMU.

TASK ADDRESS SPACE IN SYSTEM WITH BASIC MEMORY MANAGEMENT UNIT

The following points should be noted when using Figure 4-6.

1. The layout of memory is logical, not physical.

2. The layout applies only to this example; it is possible to generate systems whose layout is different from that shown in Figure 4-6.

3. The segments available to you for your bound units are 6 through F. If the group system space requirements are less than or equal to 128K-bytes, segment 3 can be used. If the group work space requirement is less than or equal to 128K bytes, segment 5 may be used.

4. One copy of segments 0.0 through 1 exists in the system in this example. These segments contain the system global space. All tasks in the system can access these segments.

5. Segments 6 through F are unique to the task unless they are being shared. If one of these segments is being shared, each task sharing the segment accesses the same copy of the segment. When a segment number is assigned by the Memory Manager, the lowest available segment (or segments for objects of size greater than 128K bytes) beginning with the group work space segment (GWS) plus 2 (segment 6 in this example) will be used. If all segments from GWS+2 through large segment F have been used, the segments GWS+1 (segment 5 in this example) and GSS+1 (segment 3 in this example) are allocated in that order, if available.

6. Only one copy of the group work space segment (segment 4 in this example) exists per task group. All tasks in the task group have unlimited access to this segment. Only one copy of the segment that contains group system space (segment 2 in this example) exists per task group. All tasks in the task group have read and execute access to this segment. Both the group work space and the group system space segments are dynamically expanded as demands are made on them. Each space can grow to a maximum of 256K bytes if the adjacent ascending segment descriptor (segment 3 for the group system space and segment 5 for the group work space in this example) has not previously been allocated to contain a task private segment.

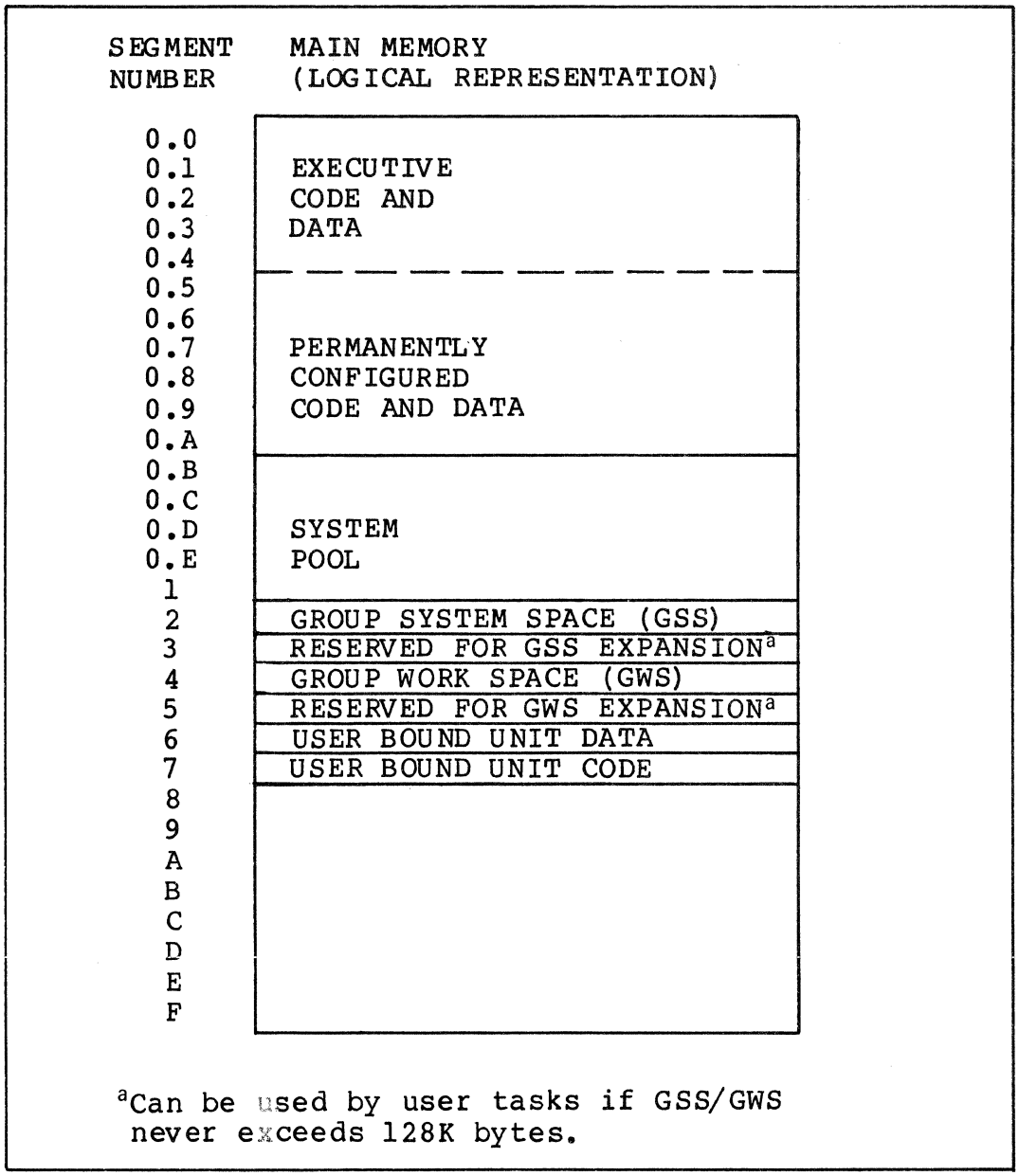


Figure 4-6. Task Address Space in BMMU System

TASK ADDRESS SPACE IN SYSTEM WITH EXTENDED MEMORY MANAGEMENT UNIT

The following points should be noted when using Figure 4-7.

1. The layout of memory is logical, not physical.

2. The layout applies only to this example; it is possible to generate systems whose layout is different from that shown in Figure 4-7.

3. The segments available to you for your bound units are 80 through FF.

4. One copy of segments 00 through 7F exists in the system in this example. These segments contain the system global space. All tasks in the system can access these segments.

5. Segments 80 through FF are unique to the task unless they are being shared. If one of these segments is being shared, each task sharing the segment accesses the same copy of the segment. When a segment number is assigned by the Memory Manager, the lowest available segment (or segments for objects of size greater than 128K bytes) beginning with segment 80 will be used.

6. Only one copy of the group work space segment (segment 46 in this example) exists per task group. All tasks in the task group have unlimited access to this segment. Only one copy of the segment that contains group system space (segment 44 in this example) exists per task group. All tasks in the task group have read and execute access to this segment. Both the group work space and the group system space segments are dynamically expanded as demands are made on them. Each space can grow to a maximum of 256K bytes if the adjacent ascending segment descriptor has not been previously allocated to contain a task private segment.

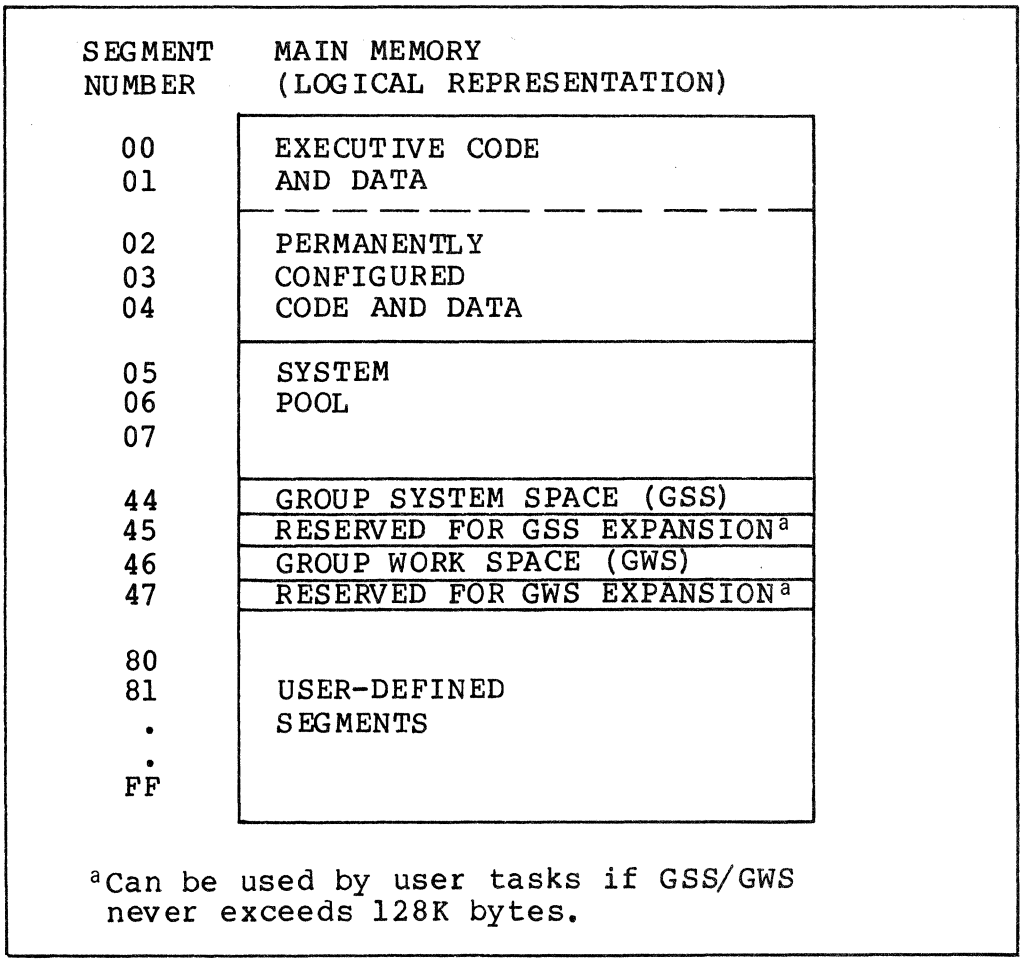


Figure 4-7. Task Address Space in EMMU System

Section 5
TASK EXECUTION

A task can be characterized as the execution of a sequence of instructions that has a starting point and an ending point, and performs some identifiable function. A task can initiate another task for execution or terminate itself by calling the task management commands or macrocalls. Multiple tasks can operate independently of and asynchronously to each other.

Each application, system, or device driver task operates at an interrupt priority level, one of the 64 priority levels provided for each central processor by the hardware and firmware. This section describes the processing of priority levels, including context saving of interrupted tasks and the assignment of priority levels and logical resource numbers to tasks. This section also describes task communication and coordination as well as deferred processing.

CENTRAL PROCESSOR INTERRUPT PRIORITY LEVELS

All system tasks, device drivers, and application tasks are assigned interrupt priority levels that indicate the order of their execution. This order of execution may be changed due to timeslicing (see below) or because this is a multiprocessor system.

Control of the central processor is given to the highest active interrupt level. However, in multiprocessor systems, a task at the higher priority may execute at the same time as a task of the lower priority since each task is executing on a different central processor.

Each central processor provides 64 potential interrupt priority levels that are used by the hardware to order the processing of events. These levels are numbered from the highest priority (level 0) to the lowest priority (level 63). Levels 0 through 5 are reserved. Level 63 is the "system idle" level. The intervening levels (6 through 62) are assigned to logical resources (that is, devices and tasks).

The determination of which priority level is to receive central processor time is based on a linear scan of the level activity indicators. The level activity indicators are maintained by the hardware in four contiguous dedicated memory locations in each central processor (see Figure 5-1). Each bit that is "on" denotes an active priority level; each bit that is "off" denotes an inactive level.

Bits can be set "on" by software or by hardware events (interrupts). Most interrupting hardware devices are associated with priority levels during system configuration (by directives in the CLM_USER file). The three highest priority levels have dedicated assignments of special hardware/firmware functions such as incipient power failure, watchdog timer runout, and trap save area overflow. Priority level 3 is reserved as an inhibit level, level 4 is reserved for internal system use, and level 5 is dedicated to the real-time clock. Succeeding levels are user-configurable as device levels. Following these are two levels reserved for system use. Except for level 63, the remaining levels can be used for application tasks. Level 63 is reserved for an always active software idle loop or, in multiprocessor systems, for the task dispatcher.

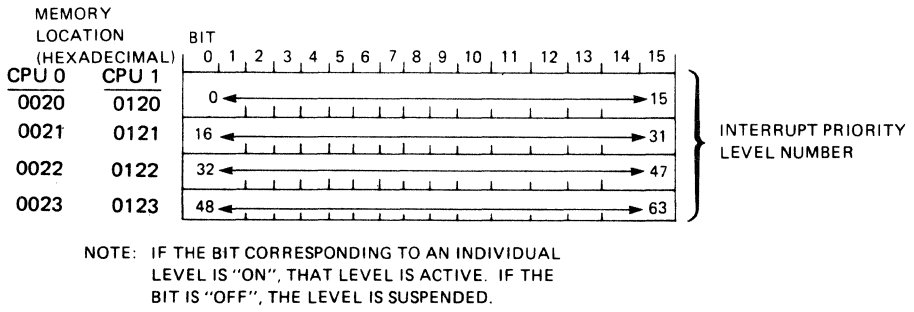


Figure 5-1. Format of Level Activity Indicators for each Central Processor

When a given priority level is the highest active level, it receives all available central processor time until it is interrupted by a higher priority level or until it relinquishes control by suspending itself (setting its level activity indicator off). If a priority level is interrupted by a higher priority level, its level activity indicator remains on and it will resume execution of the interrupted logical resource when it again becomes the highest priority level. Each time a priority level change occurs, the hardware/firmware saves the central processor context of the task running at the previously highest active level and restores the central processor context of the task running at the new highest active level. Interrupting a task, saving the context of a task, selecting and starting the highest priority level task, and restoring the context of a task are done without software involvement.

INTERRUPT SAVE AREA

The context of a level (task) can include the contents of the program counter, S-register, B-registers, I-register, K-registers, R-registers, M-registers, SIP registers, and CIP registers. The context is stored for each central processor in a block of memory known as an Interrupt Save Area (ISA). The hardware/firmware context save/restore function finds the appropriate ISA through a pointer supplied in the interrupt vector for that level. The interrupt vectors are a set of contiguous memory locations containing an entry for each potentially active priority level and ordered by ascending priority level number. Figure 5-2 illustrates the order of the priority levels, their corresponding interrupt vectors, and the format of an ISA. *

TASK DISPATCHING

The way in which a task receives central processor time depends on whether the system has one or more than one central processor.

In a monoprocessor system, tasks are dispatched according to their priority level. The task at the highest priority level receives all available central processor time until it is interrupted by a task with a higher priority level or until it suspends itself. In a multiprocessor system, all tasks are dispatched from a general ready queue. The tasks are placed in the queue according to their priority level, with higher priority tasks at the top of the queue. The level at which a task executes stays the same, but the central processor in which it executes may vary.

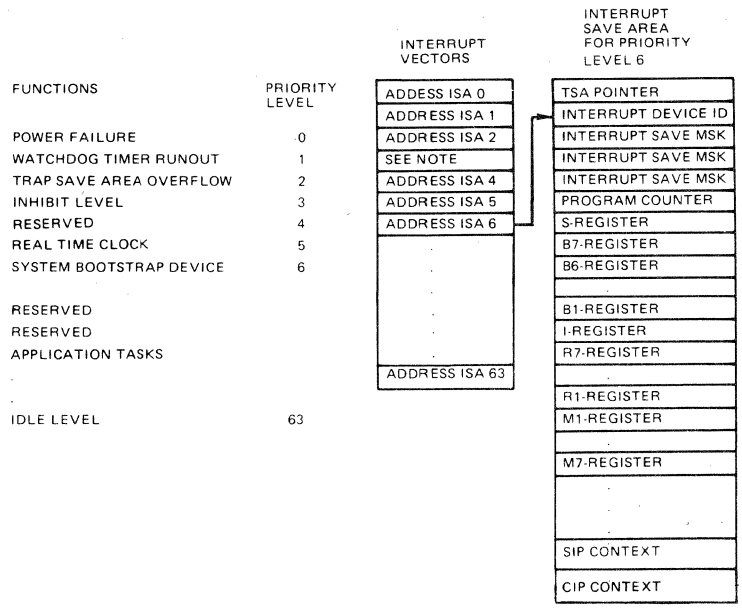


Figure 5-2. Order of Interrupt Vectors and Format of Interrupt Save Areas for Each Central Processor

Monoprocessor Task Dispatching

When a task in a monoprocessor system is at the highest active priority level, it receives all available central processor time until it is interrupted by a task at a higher priority level, until it relinquishes control by suspending itself, or until it has control taken away from it due to timeslicing (see below). If a task is interrupted by a higher priority task, it will resume execution when it again becomes the task at the highest priority level.

When more than one task is assigned the same priority level, a system software task at a higher level regulates in round-robin fashion the sharing of the level between tasks. (Timeslicing may put the task at the end of the round robin queue or demote it to a lower priority.) Thus a task does not block a level when the task is put in a waiting state after a request to wait, wait on list, request semaphore, or terminate, or after a system service macrocall that does a wait for a data transfer. The context of another task on the same level will be linked to the level interrupt vector instead (refer to "Timeslicing" below).

Multiprocessor Task Dispatching

In a multiprocessor system, the Executive maintains a queue of ready tasks ordered by priority level. This queue is called the general ready queue. The Executive dispatches the task at the top of the queue whenever a central processor becomes free to provide service. A dispatcher task runs at level 63 in each central processor and dispatches a task whenever it receives central processor time.

The dispatcher tasks attempt to balance the load so that high priority tasks are serviced before low priority tasks and all processors are used as fully as possible.

TIMESLICING

The technique of timeslicing minimizes the ability of user tasks that use large amounts of central processor time to interfere with interactive users of the system. In DPS 6 systems, timeslicing uses the Real-Time Clock Interrupt Servicing task (which executes at level 5) to check all tasks at a configured user level and below. Configuration of timeslicing is automatic. All user levels execute in a timesliced manner. Timeslicing options are discussed in the System Building and Administration manual.

The way timeslicing operates differs according to whether the system is monoprocessor or multiprocessor.

Monoprocessor Timeslicing

At each clock interrupt, a check is made to see if the task at the highest active user level has exceeded the configured value for a timeslice. (The system builder may specify the length of a timeslice in milliseconds or may accept the system default.) If the execution of that task has exceeded the timeslice value without waiting for some event, the task is removed from the front of the queue for its priority level and is placed at the end of that queue.

If a configured number of timeslices occur without the task waiting on any event, the task is demoted one priority level (that is, the task's priority level is increased by one). The task can be demoted again and again until it has been demoted the configured number of levels or has reached priority level 62.

Each time a task that was demoted waits for some event, it is promoted one level (that is, its priority level is decreased by one). The task can be promoted again and again until it reaches its assigned priority level.

Multiprocessor Timeslicing

At each clock interrupt, a check is made to see if the task at the highest active user level has exceeded the configured value for a timeslice. (The system builder may specify the length of a timeslice in milliseconds or may accept the system default.) If the execution of that task has exceeded the timeslice value without waiting for some event, the task is placed on the general ready queue as the last entry among tasks of its priority.

If a configured number of timeslices occur without the task waiting on any event, the task is placed on the ready queue and demoted one priority level (that is, the task's priority level is increased by one). The task can be demoted again and again until it has been demoted the configured number of levels or has reached priority level 62.

Each time a task that was demoted waits for some event, it is placed on the ready queue and promoted one level (that is, its priority level is decreased by one). The task can be promoted again and again until it reaches its originally assigned priority level.

TRAP HANDLING

The hardware provides a means by which certain events that occur during the execution of a task can be "trapped", with control being passed to software routines designed specifically to cover the condition causing the trap. Events such as the execution of a MOD 400 monitor call, or the detection of a program error, hardware error, arithmetic overflow, or uninstalled optional instruction cause traps (control transfers to designated software routines) to occur.

Traps are divided into two classes: (1) standard system traps, for which routines are supplied with the system, and (2) user-specific traps, for which users supply their own handlers.

An application program can designate which traps are to be handled by using the enable/disable user trap macrocalls (refer to the System Programmer's Guide - Volume II for details). If an enabled trap occurs in the user program, the Trap Manager transfers control to the connected trap handler for the condition causing the trap. A trap that is enabled is local to a task. Such a trap neither affects nor is affected by the handling of the same trap in another task, even within the same task group.

Any trap that occurs when its handler is not enabled, or that does not have a handler to process it, causes the executing task to be aborted.

SYSTEM FEATURES AFFECTING TASK EXECUTION

While MOD 400 does monitor resource use within a task group and among task groups, tasks and task groups must cooperate in their use of system resources to ensure smooth operation of the application.

Priority Level Assignments

Priority levels 6 through 62 are available for assignment to system, device driver, and application tasks. The system builder establishes the priorities of system tasks and driver tasks during configuration. (On the DPS 6/22, the Autoconfigurator establishes these priorities.) You assign the priorities of application tasks when you create task groups. Priority levels with low numeric values have higher priority than those with high numeric values. The procedures for establishing priorities are described below.

ASSIGNING PRIORITY LEVELS TO DEVICES AND SYSTEM TASKS

The system builder specifies hardware interrupt priority levels through an argument of the Configuration Load Manager (CLM) DEVICE directive. (The Autoconfigurator is used on the DPS 6/22.) When the system builder specifies a particular type of device, the appropriate Honeywell-written device driver is loaded as part of the system. The two priority levels following the last one assigned to a configured device are used by system tasks and cannot be assigned to application tasks.

One example of priority level assignment is shown in Table 5-1. Levels 0 through 5 are assigned by the system and are not available to any user. The operator terminal is assigned to level 8; however, the system builder can assign any appropriate level to the operator terminal through a DEVICE directive. (If the operator terminal is connected to the system through a communications controller, it must be at a lower (numerically higher) level than the Communications Supervisor.) At initialization, the system bootstrap device is assigned to level 6. This assignment remains in effect unless changed by a DEVICE directive.

Peripheral devices may be assigned to levels on both central processors in a multiprocessor system. This assignment is done automatically by the system.

Table 5-1 indicates Input/Output (I/O) devices, and not device drivers, to stress that each peripheral device must have at least one level assigned to it. Except for communications devices, peripheral devices cannot share a level. If there are two printers, each must be assigned a unique level even though there is only one copy of the associated I/O driver. Communications configurations require at least one nonsharable level dedicated to processing communications interrupts. This level must be higher than any level assigned to a communications device.

Communications devices can share a level. For example, four teleprinters (TTYs) and one Visual Information Projection (VIP) terminal can be configured to share one level or to use up to five levels. The priorities in Table 5-1 provide maximum throughput because devices with high transfer rates are assigned higher priorities than devices with low transfer rates.

Theoretically, the system builder could assign a level number as high as 59 to a device. In this case, levels 60 and 61 would be used by the system and only level 62 would be available for user task groups. In practice, however, the system builder would want to reserve more than one level for user task groups, especially for a system with a large number of devices. If priority levels 6 and 7 are assigned as shown in Table 5-1, the theoretical range of levels assignable through CLM COMM directives is 8 through 58. For a device associated with a COMM directive, the range is 9 through 59.

Table 5-1. Sample Priority Level Assignments for Tasks and Devices

Physical Priority Level	Base Priority Level	Use	Comments
0 1 2 3 4 5	N/A N/A N/A N/A N/A N/A	Power failure handler Watchdog timer runout TSA overflow Inhibit interrupts Reserved Real-time clock	Levels 0 through 5 are automatically assigned by the system.
6	N/A	System bootstrap device	Set to level 6 at system initialization but can be changed.
7	N/A	Communications Supervisor	Must be higher level than any communications device.

Table 5-1 (cont). Sample Priority Level Assignments for Tasks and Devices

Physical Priority Level	Base Priority Level	Use	Comments
8	N/A	Operator terminal	Can be assigned any available level.
9 9 9	N/A N/A N/A	TTY device TTY device TTY device	Communications devices can share priority levels.
10 10	N/A N/A	Removable cartridge disk Fixed cartridge disk	The priority level for a pair of fixed/removable disks must be the same.
11 12 13	N/A N/A N/A	Diskette Diskette Diskette	
14 15	N/A N/A	Line printer Card reader	
16 17	N/A N/A	Reserved by system Reserved by system	The two levels following the last device-assigned level are used by the system.
18 19 . . . 	0 1 . . . 10	Task group A Task group B . . . Task group n	
. . .			
63	N/A	System idle loop or task displatcher	Always active.

ASSIGNING PRIORITIES TO APPLICATION TASKS

You assign priorities to user task groups and tasks when you create or spawn them. The command to generate a task group contains an argument that specifies the base priority level for the task group. The base priority level is relative to the highest number priority level assigned to a configured device. When a task group is assigned a base priority level of zero, the lead task of the group executes at the physical interrupt priority level that is three level numbers above the highest level number assigned to a configured device. 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 level at which a task executes is the sum of the following:

1. The highest level number assigned to a configured device plus 3
2. The base priority level number of the task group
3. The relative priority level of the task within that group.

This sum must not exceed 62.

Interactive user tasks are usually given higher priorities (lower level numbers) than absentee user tasks. Tasks that are I/O bound should be run at a higher priority than tasks that are central processor (CP) 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 CP-bound tasks. Otherwise, if the CP-bound tasks have a higher priority, the I/O devices would be idle while I/O-bound tasks waited to receive central processor time. (Timeslicing minimizes the ability of CP-bound tasks to interfere with interactive and I/O bound tasks.)

Logical Resource Number

A logical resource number (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

The system uses DEVICE directives to assign LRN values. Device LRNs may have values from 0 through 252, and from 256 through 4095. Figure 5-3 is an example of LRN and priority level assignments for devices and system tasks.

LRN	Level	Use
	0	
	1	
	2	
	3	Inhibit interrupts
	4	
	5	Real-time clock
0	6	Operator terminal
1	7	Disk
3	8	Line printer
4	9	Serial printer
5	10	Card reader

Figure 5-3. Example of LRN and Priority Level Assignments for System Tasks and Devices

APPLICATION TASK LRNs

LRN assignments to application program tasks within each task group are not dependent on the system configuration on which the application task group is running. You can assign LRNs or have the system select the highest numbered LRN available at task creation. LRNs are assigned to task code within an Assembly language application program through specification of the Create Group and Create Task macrocalls as well as the macrocalls that build data structures ($IORB, $TRB, and so forth). LRNs can be assigned at the control language level through the commands for the creation of task groups and tasks. An LRN for an application task can have any value from 0 through 4095. Within a task group, the LRN for each task must be unique. More than one LRN can be associated with the same priority level (for example, two tasks at level 23 can have LRNs of 28 and 29, respectively).

Two kinds of tasks do not have LRNs:

- The lead task of any task group
- Any spawned task.

Logical File Numbers

Logical file numbers (LFNs) are internal file identifiers associated with file pathnames at the source language program level or at command level. LFNs can be associated with file pathnames in Assembly language or COBOL programs, or through Create File, Get File, and Associate commands. LFNs can be used to reduce program dependence on actual file pathnames (which are likely to vary). An LFN can have any value from 0 through 4095.

Task and Resource Coordination

Tasks can be coordinated in either of two ways:

- Through the use of tasking requests
- 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 can use it to 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. Use of semaphores in an application is essential if the application has multiple tasks and is sharing data in memory.

For each resource to be controlled, you define a semaphore and give it a 2-character (ASCII) name. The semaphore name is a system symbol recognized by the system control software; it is not a program symbol that needs Linker resolution. The agreement is that each requester of a resource whose use must be coordinated issues appropriate system service macrocalls to the named semaphore to request or release the resource. The task that defines the semaphore assigns the semaphore's initial value. The system control software maintains the current value of the semaphore so as to coordinate requesters of the resource being controlled. A requester obtains use of a resource if the semaphore value is greater than zero at the time of the request. If the value is zero or negative, the requester is either suspended (waiting for the resource) or notified that no resource is available, depending on how the request was made.

System service macrocalls are used to:

- Define a semaphore and give an initial value ($DFSM).
- Reserve a semaphore-controlled resource ($RSVSM). This macrocall subtracts a resource or queues a waiter for the resource (that is, it decrements the current-value counter). $RSVSM suspends the requesting task until the resource is ready.
- Release a semaphore-controlled resource ($RLSM). This macrocall adds a resource or activates the first waiter on the semaphore queue (that is, it increments the current-value counter).

- Request the reservation of a semaphore-controlled resource ($RQSM). This macrocall queues a request block (SRB) if the resource is not available (that is, it decrements the current-value counter). The requesting task must test the queued SRB subsequent to the request in order to determine when the resource is granted. The requesting task continues excuting until it executes a $RSVSM macrocall; then it waits.

- Delete a semaphore ($DLSM).

A semaphore is a gating mechanism. The initial value you give to it depends on the type of control you want to exercise. For example, assume that you want to restrict access to a particular resource to a one user at a time. The mechanism would work in the following way:

1. Task A defines a semaphore by issuing the macrocall:

   ```
   $DFSM ZZ
   ```

   Omission of the value argument causes the initial value to be set at 1.

2. Task B now issues a $RSVSM macrocall. The counter is decremented to 0. Task B gets the resource for itself, knowing that no other task using the semaphore mechanism is now using or can obtain the resource.

3. Task C issues a $RSVSM macrocall. The counter is decremented to -1. Task C is suspended and put on the semaphore queue in first-in/first-out order (because Task B is still using the resource).

4. Task B issues a $RLSM macrocall when it finishes with the resource. The counter is incremented to 0. Task C now gets the resource. After Task C issues the $RLSM macrocall, the value again becomes 1.

Use of resources by more than one user at a time can be arranged by adjusting the initial value of the semaphore. For example, an initial value of 2 allows two users, a value of 4 allows four users, and so on. The value chosen as the initial value of the semaphore depends on the nature of the resource and its intended use.

If it is undesirable for a task to be suspended while a resource is in use, the $RQSM macrocall can be used instead of $RSVSM to reserve a resource. $RQSM is an asynchronous reservation request ($RSVSM is a synchronous request) that causes a request block to be queued for the resource so that the issuing task can do other processing before the needed resource is available.

TASK HANDLING

More than one task can be concurrently active under MOD 400. In a multiprogramming environment, a task in each of several task groups can be active and compete for system resources. Another possibility is a multitasking application where several tasks executing under one task group can be active to compete for system resources among themselves and with tasks from other task groups. A FORTRAN or Assembly language program can include requests to activate several tasks and synchronize their execution; these requested tasks can execute concurrently. A COBOL, BASIC, C, Pascal, or Ada program executes as a single task, but can include commands to activate other tasks.

For the system to sequence the execution of tasks, each task must be assigned to a priority level. In monoprocessor systems, task competition for the central processor resource is governed by the hardware/firmware linear priority scan of level activity indicators. Tasks on the same priority level execute serially in the order in which they are requested. In multiprocessor systems, tasks are ordered in a software queue according to their priority levels. The task at the top of the queue is dispatched when a processor becomes free. When it is assigned to a processor, the task excutes at the same priority level as it would on a monoprocessor system.

The highest priority active task receives all available central processor time until it waits, exceeds the timeslice value, terminates, or is placed in hold. In both monoprocessor and multiprocessor systems, the task could be interrupted by a higher priority task.

It should be noted that all device drivers are considered to be tasks in the above sense. Using the File System, buffered device drivers can execute concurrently with tasks. Drivers execute on the central processor priority levels assigned to individual devices and thus have their own contexts. The device drivers provided in the system are written in reentrant code, are capable of servicing multiple devices, and execute on any central processor in a multiprocessor system.

A user task becomes active when a Spawn Task or Enter Task Request command is issued for it. The Spawn Task command can request that the invocation of the task be delayed until a specified time interval has elapsed. FORTRAN programs can cause a task to become active through the START and TRNON statements. Assembly language programs can issue a $RQTSK OR $SPTSK macrocall to activate a user task. Any application program can issue a command to spawn or request a task by calling the ZXEXCL run-time routine.

To terminate, tasks of Assembly language programs must contain a Request to Terminate ($TRMRQ) macrocall. Compilers provide this call in the object text. $TRMRQ is executed after the task completes execution.

When you want more than one task to execute concurrently, you must specify each task in a Create Task or Spawn Task command (or system service macrocall).

The procedural code for a requested task is either in a unique bound unit or in a bound unit shared with a task that was previously created. When a task is requested, the system searches for its identifying LRN in the table of LRNs associated with the task group under which the task is executing. The system activates the task if it is not already active.

TASK STATES

Tasks can exist in any of the following logical states:

- Dormant. There is no current request for the task. A task enters the dormant state if it is created but never requested or a terminate request is issued against it. A task remains dormant until a request is placed against it or it is deleted. If deleted, it is erased, memory is reused, and the task cannot be reactivated.

- Active. A task is executing or ready to execute when its priority level becomes the highest active level in the central processor. A task remains active until it waits, terminates, or is suspended. Tasks in the general ready queue are active.

- Wait. A task is not executing because it may have caused its own execution to be interrupted until the completion of an event such as the completion of a requested task, or until a timer request is satisfied, or until a task releases a semaphore. A waiting task loses its position in the priority level round-robin queue.

  An I/O order to disk, magnetic tape, the operator terminal, or an unbuffered card reader usually results in a wait condition. Task code written in FORTRAN or Assembly language will also wait in the following circumstances: (1) a write order is issued to an interactive terminal or to a printer when a previous write has not completed, (2) a read order is issued before the transfer of the current message from an interactive terminal is complete (the RETURN key is not pressed). In COBOL, these circumstances result in a wait if the program is executing its I/O statements in synchronous mode; otherwise, if in asynchronous mode, a status code value of 9I is returned with no waiting.

- Suspend. A task is removed from execution by an external human action (for example, the operator entering a Suspend Group command or a user interrupting a program with a Break action). The task is activated through another human action (for example, the operator enters an Activate Group command or a user enters a command after the Break action).

EXAMPLE OF SYSTEM INTERACTION WITH USER TASKS

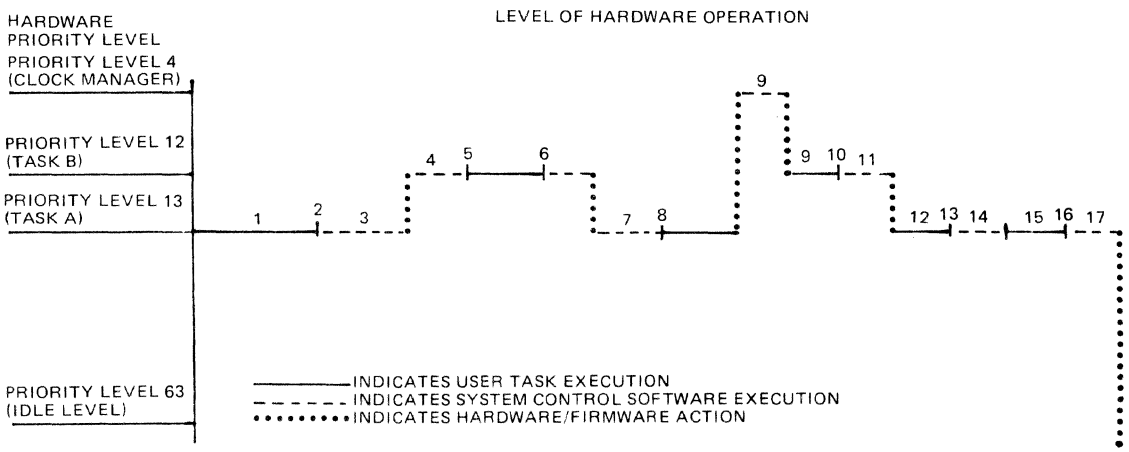
Figure 5-4 illustrates a typical interaction between the system control software and two tasks within a group. In this example, Task A has an absolute priority level of 13 and Task B has an absolute priority level of 12. The absolute priority level is obtained by adding a task's relative priority level, the task group's base priority level, and the highest system physical priority level plus three.

Figure 5-4 indicates the priority levels at which the central processor runs as the sequence of events occurs. The diagram also indicates the consecutive activity of user tasks, the system control software, and the hardware and firmware. The numbers in the diagram correspond to the numbers in the sequence of events and are explained in order in the text.

Note that if this were a multiprocessor system, Task B might have been dispatched to a processor other than the one on which Task A was running.

OPERATOR TERMINAL I/O LOGGING

If desired, you can create a disk-based log that captures all traffic involving the operator terminal. Each message sent from or received at the operator terminal is recorded in this log with a time stamp as a separate log record. The logging mechanism maintains two files and, when one is filled, automatically switches to the other. For more information, refer to the System User's Guide.



User Task Execution	System Control Software Execution
1. Task A is running; task B is requested by task A, or entered or spawned through a command.	
2. Task A does not issue a wait.	
	3. Places the request in the request queue for task B. (Assume that there are no other requests in this request queue.) Activates priority level 12.
	4. Examines the request and ascertains task B's starting address from start address data accompanying the request.
5. Task B begins execution because its priority level is higher than that of task A.	
6. Task B issues a call to the clock manager and issues a wait function to wait for clock timeout. Task B is suspended.	
	7. Now operating at the priority level of task A, the highest active priority level, returns control to task A.
8. Task A resumes execution.	
9. Task B's clock-related wait times out. The clock manager interrupts task A. Task B's priority level is activated.	10. Task B resumes execution and continues to completion, then issues a terminate call.
	11. Removes the request from the request queue for task B. Suspends priority level 12.
12. Task A resumes execution because its priority level is now the highest active level.	
13. In a multitasking program, task A could issue a wait call to wait for completion of task B.	
	14. Detects that task B's request is marked as terminated. Control is immediately returned to task A.
15. Task A continues to completion and issues a terminate call.	
	16. Removes the first request from the request queue for task A. If there are no additional requests in this request queue, suspends priority level 13. If there are no remaining active priority levels, idles at priority level 63.

86-025

Figure 5-4. System Interaction with User Tasks in a Monoprocessor System

INTERTASK AND INTRATASK GROUP COMMUNICATION

Information can be passed among task groups and tasks by means of request blocks, common files, and the message facility.

Request Blocks

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. The request blocks can contain data or pointers to information structures. All request blocks must be in common address space so that they can be shared by the tasks. (Refer to the System Programmer's Guide for details on building request blocks.) Higher level languages cannot use request blocks directly; they require called subroutines written in Assembly language.

Common Files

Tasks within the same task group and tasks within different task groups can communicate through disk files. The concurrency status must be the same for all tasks using the files. The requesting tasks must have access rights to the files.

A pipe is a special type of sequential file that also provides synchronization and queuing facilities to cooperating tasks. Pipes are used by tasks in different task groups (applications) or in the same task group to communicate with each other.

Message Facility

The message facility allows two or more task groups (users) or two or more tasks within a task group to communicate with one another. This communication is done through containers called "mailboxes." Messages (requests) sent to a task or task group are queued in a mailbox and are dequeued when received.

To control the sending and receiving of messages, the message facility provides a number of macrocalls and commands. One set of macrocalls (Initiate, Send, and Terminate Message Group) allows a message (a request) to be sent to a mailbox; another set of macrocalls (Accept, Receive, and Terminate Message Group) allows a message to be received from a mailbox. Commands are provided to allow you to send, receive, list, and cancel messages (requests). The Mail command is provided to allow you to send messages (mail) to another user's mailbox and to display mail in your own mailbox.

Deferred processing of print and task group requests is carried out through the use of the message facility. Deferred processing is described later in this section.

Before the message facility commands or macrocalls can be used, and before the deferred processing of print and task group requests can be initiated, you (or the operator) must create the mailboxes and activate the message facility task.

The paragraphs below describe mailbox creation, the activation of the message facility task, and the command and macrocall interfaces.

CREATING MAILBOXES

Three steps are involved in the construction of a mailbox. You must create the mailbox root directory, create the mailboxes, and set access controls on the created mailboxes. (Refer to the Commands manual for details.)

The mailbox root directory is the directory that is to contain the simple names of the mailboxes.

The system assumes that the mailbox root directory is in the MDD directory. (An MDD directory is supplied on the system root volume.) You, however, are free to create your own mailbox root directory through the Create Directory command.

Each mailbox is created through the Create Mailbox command. This command creates a directory corresponding to the mailbox name and a file ($MBX) within that directory defining the mailbox attributes.

To prevent unauthorized use of the message queues, you should set access controls as follows:

- Senders must be given list access on the directory defining the mailbox.
- Receivers must be given read access on the $MBX file for a given mailbox.

Individual mailboxes can be deleted through Delete Mailbox commands.

ACTIVATING MESSAGE FACILITY TASK

The Start Mail operator command activates the message facility. This command contains an optional argument used to set the name of the mailbox root directory to other than the default directory pathname (>>MDD).

MESSAGE FACILITY COMMAND INTERFACE

The commands that can be used to send/receive messages (mail) are Mail (MAIL), Send Message Mailbox (SMM), and Accept Message Mailbox (AMM). Commands are also provided to list and delete messages.

Mail Command

The Mail command (also referred to as the local mail facility) is used to send and receive multiline messages to/from the mailbox whose name (id) is the same as the person id of the receiving user. A message sent by a Mail command is queued in the mailbox and displayed only if the receiving user issues a Mail command.

To send a mail message, you issue the Mail command, specifying the mailbox id (person id) of the user to receive the messages. The message to be sent can be located in a file (named by an argument of the command) or it can be entered after the Mail command has been executed.

To receive mail messages, you issue the Mail command without arguments. The contents of your mailbox are displayed when the command is executed. If you request deletion of the messages, they are deleted from the mailbox after being displayed. Otherwise, the messages remain in the mailbox.

Send Message Mailbox and Accept Message Mailbox Commands

The SMM and AMM commands (also referred to as the local message facility) are used for single-line messages that must be viewed immediately or at a specified time. The AMM command is used to specify that messages sent by the SMM command be displayed when received or at the time specified in the SMM command.

To send a message, you issue the SMM command, specifying the person id (mailbox) to which the message is to be sent. The message is included in the command line. You can include the -TIME argument to specify a delivery time for the message. You can send a broadcast message by specifying * in place of the mailbox in the SMM command. The message will be sent to all logged-on users who have issued an AMM command indicating their willingness to accept messages.

To receive messages, you issue the AMM command. Messages already in your mailbox are displayed. Subsequent messages are displayed when placed in your mailbox. Messages whose date and time for display have not been reached are not displayed. Messages are deleted from your mailbox as soon as they are displayed.

Senders can use both the Mail and SMM commands. A receiving user who issues a Mail command receives both types of messages. If you issue an AMM command, you receive only messages sent by the SMM command unless you specified the -IMBX * argument. Only when this argument is specified will you receive both types of messages.

The -IMBX argument also allows you to specify by name the sending user from whom you will accept messages for immediate display. Messages sent by other senders are stored in your mailbox. The -AMBX argument allows you to obtain messages from mailboxes other than the one associated with your person id.

MESSAGE FACILITY MACROCALL INTERFACE

You can use the message facility on the Assembly language level by using the macrocall interface. To permit the sending and receiving of messages, the message facility provides the following macrocalls:

- Initiate Message Group ($MINIT)
- Send ($MSEND)
- Accept Message Group ($MACPT)
- Receive ($MRECV)
- Terminate Message Group ($MTMG)
- Count Message Group ($MCMG)
- Cancel Enclosure Level ($MCME).

The information associated with these macrocalls can be passed by means of request blocks.

A task group that wishes to send a message to a mailbox must issue a $MINIT macrocall to open the send-message session. The mailbox is identified by a name entered in the request block. As a result of this macrocall, the message facility returns a message id, unique to the task group, to identify the message to the other macrocalls (that is, the send). The task group then issues one or more $MSEND macrocalls to send message data. The send-message session is closed by the $MTMG macrocall or, alternatively, by the $MSEND macrocall. The sending task group can issue the $MCME macrocall to delete the last record of an incomplete quarantine unit or the entire incomplete quarantine unit. (A quarantine unit is the smallest amount of transmitted data available to a receiving task group.) Receipt of the message can be deferred by the sender.

A task group wishing to receive a message from a mailbox issues a $MACPT macrocall to open the receive-message session. The mailbox is identified as described above for the $MINIT macrocall, and the message facility returns a message id to be used by the $MRECV and $MTMG macrocalls.

The task group then issues one or more $MRECV macrocalls to receive message data. The receive-message session must be closed with a $MTMG macrocall.

The message may be accepted on the following selection criteria:

- First available message
- Sequence number
- Submitter name
- Submitter name and sequence number.

The receiving task group can request the message in record sizes other than those in which the message was sent. The receiving task group delimits the amount of received data by range or enclosure level.

The message facility can be used most effectively by two task groups wishing to communicate if they both simultaneously send and receive a message. To accomplish this, each of the task groups should issue the $MINIT macrocall to open the send-message session and the $MACPT macrocall to open the receive-message session. In this case, the quarantine unit is the vehicle used to exchange data between the two task groups.

DEFERRED PROCESSING FACILITIES

The MOD 400 deferred processing facilities are supported by the message facility (refer to "Message Facility" above). In deferred processing, the messages are requests.

Deferring the execution of interactive and absentee task group requests makes it possible for you to gain greater control over the processing sequence. Deferring print requests allows you to obtain program independence from the availability of print devices. Queuing and later transcribing reports provides a spooling capability that places printing and punching outside of program context.

Deferring Task Group Requests

When placing an interactive or absentee task group request, you can have the request entered in a disk queue and can postpone any action being taken on the request until a specified time. When the request queue structures are on disk, memory space is conserved and the data in the queues can be recovered in the event of a system failure (refer to Section 6).

Assuming that an interactive and/or absentee task group has been created, two steps are required to defer group requests. The operator must create the request queues (mailboxes), and you must issue task group requests with optional arguments specifying the time each request is to be activated.

CREATING TASK GROUP REQUEST QUEUES

The operator uses the Create Group Request Queue command to create queue structures in which requests issued to a given task group will be stored. The operator must also issue a Start Mail command if one had not been previously issued. These procedures are described in the System User's Guide.

QUEUING TASK GROUP REQUESTS

You queue task group requests by issuing an Enter Group Request command. You can postpone action being taken on a request by specifying the -DFR (defer for interval) or -TIME (defer until date/time) arguments.

Once the operator has issued a Create Group Request Queue command for a task group, all further requests for that group are queued whether or not the requests are being deferred.

If the operator does not issue a Create Group Request Queue command, you can still submit group requests but will not be able to defer the requests.

Deferring Print Requests

The system provides a deferred printing capability under which your requests for printing specified files are queued in memory or disk mailboxes. The actual transcription of the files is done at a later time under the control of an operator-created system task group called a daemon.

After you submit a deferred print request, you can resume normal activities, log off, or reboot the system without losing the request.

The three steps involved in deferred print processing are creating the mailboxes, activating the daemon, and queuing the print requests. The information in the following paragraphs is conceptual. Detailed procedures for deferred printing are given in the System User's Guide.

CREATING PRINT REQUEST MAILBOXES

The operator establishes the mailboxes that are to contain the queued print requests. The mailboxes can be in memory or on disk. The mailbox names must be in the form $PR.Qn (n is an integer from 1 through 9 that identifies the relative priority of the queue, with 1 being the highest priority and 9 the lowest).

CREATING THE PRINT DAEMON

The operator is responsible for defining and activating the daemon to process the print requests. (The supplied START_UP.EC file creates a standard daemon task group at system startup. The operator can accept this daemon, modify it, or create his/her own.)

To create a daemon task group, the operator issues a Start Mail command (if one was not already issued), a Create Group command naming the daemon to be created, and an Enter Group Request command identifying the mailboxes to be used for queuing the requests and the devices to be used for printing.

Multiple daemon task groups can be run concurrently using common or separate sets of mailboxes and printers.

QUEUING PRINT REQUESTS

Once the daemon task group is active, you can queue print or punch requests by issuing Deferred Print commands. You can employ the -TIME argument to defer the printing of a file until a specified date and time.

Queuing and Transcribing Reports

Any file in print or punch format (i.e., any report file) can be queued and subsequently transcribed to an available printer or card punch. Report queuing and transcription is a spooling capability that provides automatic and manual report transcription, time-of-day printing or punching, and an automatic setup function that includes a sample transcription file (template).

The report queuing and transcription facilities control report transcription outside the context of the program. Reporting procedures for identical software can be totally different in different situations without requiring reprogramming.

Report queuing and transcription have three major aspects: creating a report queue, queuing a transcription request, and transcribing a report.

CREATING REPORT QUEUES

A report queue is a directory that allows you to place a report in a queue and subsequently transcribe the report. Report queues are created, modified, and deleted through Report Queue Maintenance (RQM) commands. The characteristics of the report queues are determined when the queue is created; the contents are determined when a report is placed in the queue for later transcription.

When the report queue is created, a report queue profile file is built. The report queue profile file designates the characteristics of reports that will be entered in the report queue and printed or punched at a later time. The report characteristics include:

- Name of form descriptor
- Format of reports to be queued (print or punch)
- Transcription mode (automatic or manual)
- Column number at which printing is to begin
- Line at which printing is to begin (head of form)
- Number of print lines per inch
- Number of copies of report
- Time at which report is to be transcribed
- Heading line
- Destination line.

The report queue profile file is complete when the report queue is created; however, various aspects of the profile can be overridden when the report is queued.

QUEUING REPORT REQUESTS

The name of a report to be subsequently printed or punched is placed in a report queue through the Queue Report (QRPT) command. This command also associates with the report a specialized report queue profile file that governs the details of the report transcription. Once a request has been queued, it remains queued until the file has been transcribed or the request pathname has been deleted through a report queue maintenance renew or delete function.

TRANSCRIBING REPORTS

Previously queued reports are written to a printer or card punch through the Unspool (UNSP) command. A single UNSP command can unspool all current and future reports. The printing or punching characteristics are determined by the report queue profile file created through the RQM command, the specialized report queue profile file created by the QRPT command, the user's activities, and the arguments specified in the UNSP command.

The UNSP command defines the report queue and the hard copy device to be used. After the command is executed, the specialized report file (if any) is deleted from the report queue. All reports whose profile matches the specified profile are unspooled in a single invocation of UNSP.

The report queue profile file can specify that the report is to be transcribed automatically or manually.

Automatic transcription is used when constant monitoring of a report queue is desired. When there is no transcription activity in progress, the unspool routine suspends itself for 1-minute intervals. When transcription of the queue is activated, each report in the queue is printed immediately unless one of the following is true:

- Manual mode was specified in the controlling profile.
- The specified time of day for report transcription has not been reached (or exceeded).

Manual mode is used to print reports in a nonautomated fashion. When the reports are required, the UNSP command is issued. All reports on the queue are transcribed immediately, regardless of time or mode. When the print queue is empty, UNSP terminates.

Section 6
BACKUP AND RECOVERY

MOD 400 provides facilities that enable you to backup and reorganize disk files, preserve the execution environment during a power failure, perform file recovery at the record level, and restart a program from a previously established point.

The file backup and reorganization facility consists of the save and restore functions. These functions provide you with a means of:

- Backing up disk files or volumes on 1/2-inch or 1/4-inch magnetic tapes or other media so that a copy of the files is available should the original files become corrupted.
- Consolidating (concatenating) files that have become fragmented through continuous updating.

The power resumption facility uses the memory save and autorestart unit to preserve the memory image through a power failure lasting up to 2 hours. If power is restored during this time, the power resumption facility reconnects the previously online peripheral and communication devices and restarts the tasks that were running when the power failure occurred. If the power failure lasts more than 2 hours, the memory image is destroyed and the power resumption facility disabled. When power is restored, you can reinitialize the system and use the file recovery and checkpoint facilities to restart the system from a previously established restart point.

File recovery enables you to dynamically save record images before they are updated and, if necessary, later write the images back to the file, thereby returning the file to its unaltered state. File recovery provides file integrity in the event of a system failure. MOD 400 supports file recovery through three distinct functions:

- Before-image recording - Preserves a record prior to its being updated.

- Cleanpoint or checkpoint declarations - Issued in your program to define a point at which a multirecord or multifile update transaction is complete. When an update transaction is complete, the associated before images are destroyed.

- Rollback, recovery, or restart functions - Returns the files to their unaltered state by applying all before images that have been recorded since the last cleanpoint.

File restoration procedures enable you to reconstruct disk files and/or volumes that are damaged as a result of a device failure. File restoration is provided through two distinct functions:

- After-image recording - Preserves a record of the updates made to files.

- Roll Forward utility - Reapplies updates (after images) to files to bring them up to their most recent consistent state before the device failure.

After images are used in conjunction with the Save, Restore, and Roll Forward utilities to return files to a known state if data in the files is destroyed as a result of a device failure.

The cleanpoint, rollback, and recovery functions should be used to provide file recovery in a transaction-oriented environment. They are best suited for applications in which a single transaction causes a number of record updates. In an absentee processing environment, the checkpoint and restart procedures should be used for file recovery and program restart.

The checkpoint restart facility enables you to establish a point in your program to which you can return at a later time and continue processing. The return point (checkpoint) is used to save the current status of the task group. You issue a checkpoint call in your program when you reach a point in your processing at which the program could be restarted. A restart can be performed at the most recently completed checkpoint at any time during processing. If the task group is abnormally terminated for any reason, it can be restarted at the most recent valid checkpoint.

FILE BACKUP AND REORGANIZATION

File backup and reorganization is implemented through the Save and Restore utilities. The Save utility transfers disk files and directories to 1/2-inch or 1/4-inch magnetic tape or another specified storage medium. The Restore utility reconstructs the saved files and directories and puts them back on disk. Any file that has been saved and restored is automatically reorganized for disk space and record access efficiency.

Since file access time efficiency may be lessened after a file has been in use for some time, it is recommended that disk volumes be periodically saved and restored. The files on the restored volume will be be compacted, resulting in optimal space allocation and improvements in the time required to search directories and check access rights.

Saving Files and Directories

The Save utility enables you to save an entire disk volume, a directory and all its subdirectories and files, or a specified file. If you are saving a directory, you can specify the number of levels of subdirectories (with their associated files) to be saved. Any access control lists associated with the saved files and directories are also saved, unless you specify otherwise.

The saved data, whether a whole volume, a file, or directories and files, is stored in a save file. The save file can be a magnetic tape or disk file, or an output device such as a card punch. When the Save utility processes the files and directories to be saved, it adds information that is meaningful only to the Restore utility. The saved files and directories are not just copies of the originals.

The Save utility can be executed while the files being saved are in use. Used with a journal file (refer to "File Restoration" later in this section), this type of save operation provides a dynamic and concurrent backup facility for high volume systems that cannot afford periodic shutdown to perform static file saves.

Restoring Files and Directories

You can restore from a save file all or part of the data you saved on that file. You can restore an entire volume (if you saved an entire volume), a directory and its associated subdirectories, or a specified file. Whatever you restore, you can return to the place from which you saved it, or you can place it in another directory or another volume.

Data saved from one type of disk can be restored to another type, provided the new disk has the required capacity. For example, you can restore a diskette volume onto a cartridge disk volume, or a partially filled mass storage device volume to a cartridge module disk volume.

POWER RESUMPTION

Power resumption is an optional facility that allows the system execution environment to be automatically restarted after a power interruption. The central processor must have the memory save and autorestart unit. This unit can preserve the memory image through a power failure lasting up to 2 hours. (It cannot, however, preserve the state of the I/O controllers nor can it ensure that no operational changes have been made to the mounted volumes.)

If fewer than 2 hours have elapsed when power is returned to the central processor, the power resumption facility will perform the following functions:

- Reinitialize the system software.
- Reconnect peripheral devices.
- Reconnect communication devices serviced by the Asynchronous Terminal Driver (ATD) line protocol handler or the Teleprinter (TTY*) line protocol handler. (Refer to the System Building and Administration manual and the System Programmer's Guide - Vol. I for information about these line protocol handlers.)
- Restart certain application tasks that were active at the time of the failure. Application tasks that are capable of being restarted are those using the display formatting and control facility and those containing user-written code to handle power failure and power resumption.

Implementing the Power Resumption Facility

The power resumption facility must be included in the MOD 400 Executive at system building. The central processor must contain a memory save and autorestart unit that has been activated by the operator (refer to the System User's Guide for activation procedures).

When power resumption is specified in the system building dialog, all peripheral devices and all communication devices associated with the ATD and TTY* line protocol handlers are designated as reconnectable and will be automatically reconnected when power is restored. If any ADT or TTY*-associated device is not to be automatically reconnected, the Set Terminal Characteristics (STTY) directive associated with the device must not contain the -RECONNECT argument.

Power Resumption Functions

The power resumption facility automatically performs the following functions:

- Restarts the device drivers, clock, communications subsystem, and display formatting and control facility.
- Reconnects all peripheral devices that were online at the time of the failure.
- Reconnects ATD- or TTY*-associated communication devices that were online at the time of the failure, except for those devices designated as not reconnectable.
- Restores the screen forms on reconnected terminals controlled by the display formatting and control facility.
- Resets the system date and time if the date/time clock has a separate battery backup unit.
- Reloads the memory management unit.
- Reestablishes the integrity of mounted volumes.
- Restarts application tasks that were active when the power failure occurred, provided the tasks used the display formatting and control facility or contained user-written code to handle power failure and power resumption.

If an application task is to be notified when a power resumption has occurred, it must be written to check Trap 53 when the task becomes active and is issuing its own instructions (not executing Executive instructions). (Refer to "Trap Handling" in Section 5.)

After a power resumption has occurred, peripheral devices and reconnectable ATD- or TTY*-associated devices that were online at the time of the failure are again brought online. The operator may be required to initialize certain peripheral devices. A terminal user may be required to reenter the input line if he/she had not pressed the RETURN or XMIT key when the failure occurred. (Refer to the System User's Guide for details.)

FILE RECOVERY

The file recovery facility enables you to save record images from a file before it is updated and to later write these images back to the file, eliminating the alterations made during the updating. Every time a record is updated, a copy of the record, as it exists before the update, is written to a system-created file. The system-created file is called a recovery file; the records it contains are called before images. The system uses recovery files to bring your data files to a consistent state following a software failure or a system failure such as that caused by a loss of power. When the before images are applied in reverse chronological order to your data files, the data files are rolled back to a previously established state.

Designating Recoverable Files

File recovery is optional. You designate a file as recoverable through the -RECOVER argument of the Create File command. Files not created as recoverable can be made recoverable by specification of the -RECOVER argument of the Modify File Attribute (MFA) command. Also, you can designate all files in a directory or volume to be recoverable through the Modify Directory Attributes (MDA) command. Recoverable files can be made nonrecoverable through the specification of the -NORECOVER argument in the MFA or MDA command.

Recovery File Creation

Each task group (or task in some cases) having a data file designated as recoverable has associated with it a recovery file. The recovery file is created by the system when the first before image for a recoverable file is about to be written.

All recovery files are created subordinate to your working directory, unless you specified otherwise by the Assign Recovery File command. (The names of the files are recorded in the $$CATALOG directory, which is positioned under the root directory of the system volume. This directory is maintained by the system.) Each recovery file is assigned a name of the form:

```
$$REC.nnggtt
```

where nn is the node identifier, gg is the group identifier, and tt is the task identifier.

File Recovery Process

The system recovers a data file (erases the updates made to it) by writing the before images back to the file. You declare points in your task group processing (called cleanpoints) at which all file updates are considered valid. When a cleanpoint is declared, all before images taken up to that point are invalidated. New before images are written when you again begin to update the file.

You can perform a rollback at any time during processing. When a rollback is requested, the before images are written to the file, wiping out updates made since the last cleanpoint.

Use of the cleanpoint and rollback task group functions is recommended in a transaction-oriented environment.

TAKING CLEANPOINTS

When you consider the data in your task group's file(s) to be consistent and valid, declare a cleanpoint in your task group. Cleanpoints are established by $CLPNT macrocalls in Assembly language programs and by the ZCLEAN utility in programs written in higher-level languages (for example CALL "ZCLEAN" in COBOL). When a cleanpoint is declared, the system performs the following actions:

- Writes all buffers modified by the task group to disk.
- Updates all directory records for files modified by the task group.
- Invalidates the recovery file before images that have been taken for data files used by the task group.
- Unlocks all previously locked records belonging to the task group. (Tasks waiting for these records are activated.)

The File System automatically performs a cleanpoint when a recoverable file is closed.

REQUESTING ROLLBACK

Rollback initiates the recovery of your task group's files to the condition in which they were at the last cleanpoint. If programming in Assembly language, request a rollback by coding a $ROLBK macrocall. If programming in a higher-level language, request a rollback by using the ZCROLL utility (for example, in COBOL use a CALL "ZCROLL" statement). When you request a rollback, the system performs the following actions:

- Takes before images from the recovery file and writes them to the data files used by the task group, thereby wiping out updates made since the last cleanpoint.
- Invalidates the before images for the task group's data files on the recovery file.
- Unlocks all previously locked records belonging to the task group. (Tasks waiting for these records are activated.)

The File System automatically performs a rollback when a task group terminates abnormally.

RECOVERING AFTER SYSTEM FAILURE

If recovery files exist, the operator should issue the Recover command so that the system will perform a rollback of all recoverable data files. (Refer to the System User's Guide for details.)

FILE RESTORATION

File restoration provides the ability to preserve updates that have been made to files and to apply these updates to saved versions of the files if the original versions become corrupted. You cause images of records that have been modified (after images) to be recorded in a journal (after image) file. You can then use the journal file in conjunction with the Save, Restore, and Roll Forward commands to restore files to a known state if data in the files is destroyed as a result of a device failure (if I/O errors indicate any damaged files and/or volumes, file restoration procedures are recommended).

Designating Restorable Files

You designate files as restorable by specifying the -RESTORE argument of the Create File command. Files not created as restorable can be made restorable by specifying the -RESTORE argument of the Modify File Attribute (MFA) command. Also, you can designate all files in a directory or volume to be restorable through the Modify Directory Attributes (MDA) command. Restorable files can be made nonrestorable by specifying the -NORESTORE argument of the MFA or MDA command.

It is recommended that files designated as restorable also be designated as recoverable (having the -RECOVER attribute) to provide for complete file integrity if a device or system failure occurs.

Journal File Creation

The journal file is created and maintained by the operator through the Open Journal, Close Journal, Display Journal, and Swap Journal commands. One system-wide journal file, on tape or disk, records updates made to all restorable disk files. A system-created journal history file contains the name of the current journal file and a history of all previous journal files, including the date and time they were created.

Each time a record in a restorable file is updated, the system records on the journal file the image of the record as it exists after the modification (the after image). The after image of the updated record is written to the journal file at the time the record in the file is physically updated. If the operator specifies the -BEFORE argument in the Open Journal command, the system will also record on the journal file the image of the record as it exists before the modification (the before image). You might want to record before images for audit purposes.

The journal file contains a running summary of all changes made to restorable files (for example, if a restorable file is renamed or modified, appropriate entries are added to the journal file to reflect these changes). Restorable disk files cannot be modified in any way unless the journal file has been previously opened by the operator.

File Restoration Process

For each file that is corrupted, the restoration process involves mounting a known valid version of the file, reconstructed from data preserved during a previous save operation. The save operation involves preserving the data contents and selected attributes of the uncorrupted file (by means of the Save command) before any catastrophe occurs, then restoring the file structures of the saved file (using the Restore command) after the file has been corrupted. Following these actions, you cause after images from the journal file to be applied to the restored file by using the Roll Forward command. The restored file now incorporates the changes or updates stored in the journal file since you last invoked the Save command.

File restoration offers more extensive procedures if files are corrupted following a device failure and file recovery procedures fail to return files to a consistent state.

For example, the operator opens the journal file and enters the Recover command. If the Recover command executes successfully, you can log in and continue processing. If the Recover command fails to execute successfully, the operator must close the journal file, mount saved versions of all files, and use the Restore command to reinstall the saved versions. The Roll Forward command is then entered. This command applies journal file images to all restored files, thereby updating the files to reflect modifications made after Save commands were entered for those files. File restoration is then complete and users can log in and continue processing.

CHECKPOINT RESTART

The checkpoint restart facility allows you to provide a task group file recovery and program restart capability in an absentee processing environment. Through checkpoint restart, you can establish a point in your program to which you can return at any time and continue processing. This return point (called a checkpoint) is used to save the current status of the task group request. You can perform a restart to the most recently completed checkpoint after the abnormal termination of the task group request or at any point during the processing of the task group request. A restart cannot be performed from an earlier checkpoint, nor can it be performed after the normal termination of a task group request.

Checkpoint restart does not support the use of the Listener secondary login facility.

Checkpoint

When a task requests a checkpoint, the system records the current contents of the task group's memory and the current state of tasks, files, and screen forms onto a checkpoint file you have previously assigned. The system then takes a cleanpoint for that task group so that recoverable files are synchronized with that checkpoint. (Refer to "File Recovery" earlier in this section for a description of recoverable files and cleanpoints.)

The system supports one checkpoint task and any number of other tasks that are dormant or waiting on requests placed against other tasks in the task group. (Thus, a single active command executing under the Command Processor and/or any number of nested EC files can be checkpointed.)

CHECKPOINT FILE ASSIGNMENT

You enable the checkpoint restart facility for your task group and designate where its checkpoint images are to be recorded by issuing the Checkpoint File Assignment command.

Checkpoints are written alternately to each of a pair of checkpoint files. This technique ensures the availability of the previous valid checkpoint if a failure occurs during the process of taking a checkpoint. The system locates and uses only the most recently completed successful checkpoint from the pair of checkpoint files that you specified.

When designating the checkpoint file, specify a single pathname (the last element of which can be a maximum of 10 characters). The system appends the suffixes .1 and .2 as appropriate. If the system cannot find one or both of the specified checkpoint files, it creates it (them).

TAKING A CHECKPOINT

When a checkpoint is taken, the system writes a checkpoint image and performs a cleanpoint for all recoverable files being used by the task group. If programming in Assembly language, request a checkpoint by coding a $CKPT macrocall. If programming in a higher-level language, request a checkpoint through the ZXCKPT utility (in COBOL you can use a CALL "ZXCKPT" statement or the RERUN clause in the I-O-CONTROL paragraph).

Your task group must be in a state that allows the system to take checkpoints. To be in a checkpointable state, tasks in the group can have no outstanding requests against tasks outside the group. Specifically, a task group is in a checkpointable state when each task that is part of the group has requested a checkpoint, is waiting on a request issued to another task in the task group, or is dormant (no current requests exist for the task).

Once a checkpoint is recorded by a task group, it remains available as a restart point until the next checkpoint request is completed, the current checkpoint file is disassigned (by the -DISASSIGN argument of the Checkpoint File Assignment command), or the task group request is terminated normally.

You can use the Defer Checkpoint macrocall to prevent or allow the taking of checkpoints by tasks within the task group. If you wanted to protect a procedure from being checkpointed, you would disable checkpoints at the beginning of the procedure and enable them at the end of the procedure.

The lead task of the group may be waiting on both another task that is a member of the group and a "break" request.

CHECKPOINT PROCESSING

When a task group takes a valid checkpoint, the system records the following information on the current checkpoint file established for that group:

1. Executive information including data structures, user pool memory blocks obtained by get memory operations, data segments of bound units linked with separate code and data, nonsharable bound units, and floatable overlays.

2. Status and pathnames of the standard I/O files.

3. Memory locations and pathnames of sharable bound units.

4. Current state of screen forms for terminals operating under the display formatting and control facility.

5. Status and position of all active user files (files that have been associated, reserved, or opened).

When your file information has been recorded, the checkpoint image is completed and a cleanpoint is taken. You must ensure that files to be synchronized with the checkpoint restart process have been designated as recoverable. Since the File System performs a cleanpoint when a recoverable file is closed, you may have to take a checkpoint prior to closing the file to keep checkpoint restart synchronized with the state of the recoverable file. (Temporary files cannot be designated as recoverable.)

Checkpoints cannot be taken while an active local mail message group exists. In other words, a checkpoint cannot be taken in the period between message initiation or acceptance and message termination; refer to "Message Facility Macrocall Interface" in Section 5.

Checkpoints are not made automatically obsolete by the normal termination of the task under which they were issued. To invalidate a previous checkpoint (taken during the execution of one command) before processing a new command, you must take a checkpoint immediately prior to the termination of that command.

Restart

You can perform a restart at the following times:

- During the processing of the task group request that issued the checkpoint request.
- During the processing of a task group request that was scheduled after the abnormal termination of the task group request in which the checkpoint was taken.
- When the system is reinitialized following a system failure.

When a restart request is issued, the task group issuing the request is terminated abnormally and the task group request recorded on the checkpoint file is again put into effect.

The system locates the most recently completed checkpoint and reads the checkpoint image from the file, rebuilding the Executive data structures and memory blocks, reloading bound units, and repositioning active user files.

Procedural code and workspace must occupy the same physical memory locations that were used when the checkpoint was taken. In general, task groups that are to be restarted must be the sole users of memory pools. Sharable bound units referred to by these groups must be permanently loaded (through the Load command in the system startup EC file). The configuration under which the restart is performed must be identical to that which existed when the checkpoint was taken.

REQUESTING A RESTART

To restart from the last completed checkpoint (and to abort the current task group request if restarting during the session), issue the Restart command. The operator can restart an existing task group that has a valid checkpoint by using the -GROUP argument of the Restart command. If the memory blocks required to effect the restart are not available, the restart is aborted. Specification of the -WTMEM argument of the Restart command causes the system to wait until the specific memory blocks required to perform the restart become available.

If this is a restart following a system failure, the Recover command must have been issued by the operator or through an EC file to perform a system-wide rollback of all recoverable files.

If a restart is performed during a session, the abort (termination) of the group request causes a rollback of all recoverable files in your task group. The abnormal termination of the group request causes the last completed checkpoint image to be retained as a valid checkpoint. The Abort Group and Abort Group Request commands force an abnormal termination; the Bye command causes a normal termination. (The normal termination of the Command Processor with a nonzero value in the $R2 register is treated as an abnormal termination for checkpoint file purposes.)

The Validate Checkpoint command or active function can be used to ascertain whether the specified checkpoint file pair contains a valid restartable checkpoint.

RESTART PROCESSING

When you issue the Restart command, the system performs the following steps:

1. Locates the most recently completed checkpoint.

2. Validates that the restart is being performed under the same user id as that used when the checkpoint was taken.

3. Rebuilds Executive data structures.

4. Reads nonsharable bound units, data segments, floatable overlays, and memory blocks that were obtained by get-memory operations from the checkpoint image into the same memory locations they occupied at the time the checkpoint was taken.

5. Reloads sharable bound units in the system memory pool. Only the code segment is reloaded if the bound unit was linked with separate code and data. Unless it was linked with the restart relocatable attribute (Linker RR directive), the code segment is reloaded at the same system pool memory locations occupied when the checkpoint was taken.

6. Associates, gets, opens, and positions active user files recorded on the checkpoint image. Rollback should have been performed already (refer to "Requesting a Restart" above).

7. Restores the screen content of terminals that were operating under the display formatting and control facility and were active at the time of the checkpoint.

8. Reissues the break request if such a request had been issued by the lead task at the time of the checkpoint.

9. Turns on the task that issued the checkpoint request at the next sequential instruction after the checkpoint.

The checkpointed state of the standard I/O files (user-in, user-out, command-in, and error-out) is reestablished at restart time. Modifications made to files (for example, EC files) between the checkpoint and the restart must be restricted to those that do not invalidate the repositioning of the files. A command being restarted must remain in the same position in the file; only those commands that follow the restarted command have any effect on the restarted task group request.

Sharable bound units being used by a checkpointed task group are reloaded and not restored from a checkpointed memory image (except for the data segments of bound units linked with separate code and data). Thus, all such bound units should contain only code. All sharable bound units in use by a restarting task group must be identical to the versions that existed at the checkpoint. They cannot be relinked. If an Overlay Area Table (OAT) is in use for such a bound unit, no overlay area can be reserved at the time the checkpoint is taken.

If you have application programs that issue physical I/O orders for communication devices, you must reissue connects to those devices before issuing read and write orders to them.

GLOSSARY

HT (Horizontal Tab)

Command Processor: Reserved character.

(space or blank)

Command Processor and Utilities: Reserved character; separates arguments and commands. Operator Interface Manager: At the beginning of a line, interrupts output.

! (exclamation point)

File System: A prefix indicating a physical device (sympd) name (for example, !LPT00). Line Editor: Escape character (for example, !F).

" (quotation mark)

Command Processor: Reserved character delimiting strings that contain embedded blanks (for example, "D. COOK"). See ' (apostrophe).

(number sign)

Line Editor: Signifies condition in If Data, If Range, and If Line directives. Linker: Specifies the current address.

$ (dollar sign)

Line Editor: In an address expression, represents the last line of the buffer (for example, $P). In any other Line Editor expression, represents the end of a line (for example, /DIVISION.$/). Linker: Specifies the next location (for example, BASE $). File System: First character of a macrocall name or mailbox (for example, $GTFIL).

% (percent sign)

Linker: Address argument representing the location two bytes greater than the highest address previously used in a linked root or overlay (for example, LDEF XTAG,%). Copy, Compare, Compare ASCII, and Rename Commands: Represents the character in the corresponding component and letter position of the entry name (for example, START_U%.EC).

& (ampersand)

Line Editor: Used in the string expression of the Substitute directive to indicate that the current expression is to be repeated (for example, S/TO BE/& OR NOT &/). Multi-User Debugger (numeric) and $D Debug: Address symbol, representing the next address beyond the address used in the previous debug directive. Command Processor: Reserved character. Indicates continuation of a command on more than one line. Execute Command: Indicates EC directives and comment lines (for example, &P BEGIN LINK). TCL Compiler: Indicates continuation of a statement on more than one line.

' (apostrophe)

Command Processor: Reserved character. See " (quotation mark).

() (parentheses)

Command Processor: Delimits components of an iteration set (for example, PRINT (FILEA FILEB)). Multi-User Debugger (Numeric) and $D Debug: Delimits action lines to be stored for later use. Line Editor: Delimits multicharacter buffer name; optionally, delimits single-character buffer name (for example, B(EXEC)). TCL Compiler: Indicates insertion of field value.

* (asterisk)

Line Editor, CLM, TCL Compiler: Designates an expression. Patch: Comment directive. File System: Represents one component of a file name (for example, COBPRG.*). In relation to Access Control Lists (ACLs) and Common Access Control Lists (CACLs), represents any person, account, and/or mode (for example, COOK.*.INT). List Profile Utility, Multi-User Debugger (Numeric) and $D Debug: Signifies "all."

+ (plus sign)

Line Editor: Indicates unary addition of an address (for example, +2P, 2+3). Multi-User Debugger (Numeric) and $D Debug: Performs addition.

, (comma)

Line Editor: Separates two addresses to be referenced inclusively (for example, 1,5P). CLM, Linker, Sort, and Merge: Separates arguments within directives.

- (minus sign)

Command Processor: Immediately precedes an argument (for example, -ECL). Line Editor: Indicates unary subtraction of an address (for example, -2P). Multi-User Debugger (Numeric) and $D Debug: Performs subtraction.

. (period, decimal point)

File System: (1) Separates an entry name into components (for example, COBPRG.C). (2) Used as a single element at the beginning of a pathname to indicate the working directory (for example, .>FILE_DUMP). Line Editor: (1) In an address, represents the current line of the buffer (for example, .P). (2) In an expression, requests a string containing any character (for example, /PROG.AM/). Multi-User Debugger and $D Debug: Address symbol, representing the same starting address used in the previous debug directive. TCL Compiler: Indicates the end of a statement.

/ (slash)

File System: If first character of a star name, negates the meaning of the star name (for example, /*.WORK). See * (asterisk). Line Editor: Delimits strings in Expressions and Substitute directive (for example, S/OLD/NEW/). Patch and File Change: Immediately precedes a relative location or offset. Multi-User Debugger: Separates location from repetition value. $D Debug: Separates directive from the LRN of the output device and the location from the repetition value. Linker: Precedes a comment in a Linker directive file (for example, /SECOND OVERLAY).

: (colon)

Line Editor: Indicates label definition (for example, :7).

; (semicolon)

Line Editor: Separates two addresses; the first address becomes the current line, after which the value of the second address is calculated (for example, 2;.3P). Patch: Separates arguments in Patch directives. Sort and Merge: Separates directives. Linker: Separates Linker directives on one line. Command Processor: Reserved character. Separates commands. Multi-User Debugger and $D Debug: Separates directives.

< (less-than)

File System: Indicates movement in the storage hierarchy toward the root and a change in one level in that direction (for example, <LIBRARY). Assembler and Patch: Immediately precedes a relocatable address. Multi-User Debugger and $D Debug: Specifies the condition to be satisfied in an IF directive for conditional processing of the directive line.

= (equal)

Line Editor: Print Line Number directive. Multi-User Debugger (Numeric) and $D Debug: Expresses equality for an IF directive. Linker: Address argument, specifying the base address associated with the object unit identified by an associated label (for example, BASE =OPNCRD). Copy, Compare, Compare ASCII, and Rename commands: Represents the corresponding component of a file name (for example, COPY FILE.A =.B).

> (greater-than)

File System: (1) Used at the beginning of a pathname to indicate a file or directory under the User Root Directory (URD) (for example, >SYSLIB2) and (2) Within a pathname, indicates movement in the storage hierarchy away from the root; connects two directory names or a directory name and a file name (for example, ^MYVOL>MYDIR>MYFILE). Line Editor: Go To directive (for example, >1). Multi-User Debugger and $D Debug: Specifies the condition to be satisfied in an IF directive for conditional processing of the directive line. Assembler and Patch: Indicates short displacement address.

>> (two consecutive greater-than signs)

File System: Used at the beginning of a pathname to indicate a file or directory under the System Root Directory (SRD) (for example, >>SID).

? (question mark)

Line Editor: Address prefix directive. Copy, Compare, Compare ASCII, and Rename Commands: Represents any character appearing in the corresponding component and letter position of a file name (for example, START_?P.EC). (See %.) File System and Command Processor: Immediately precedes a symbolic start address (entry point) in a bound unit name (for example, NOW?TIME). In some commands, requests help (for example, EP (Edit Profile)).

@ (at-sign)

Command Processor: Delete the previously typed character (for example, TIMM@E).

[] (brackets)

Command Processor and TCL Compiler: Delimits active functions (for example, (&P THE TIME IS [TIME]). Multi-User Debugger (Numeric) and $D Debug: Signifies the contents of the location defined by the expression within the brackets.

^ (circumflex)

File System: (1) Indicates a root directory, and must immediately precede a root directory name (for example, ^SYSRES) and (2) Used as a single element at the beginning of a pathname to indicate the root of the working directory (for example, ^>MYDIR). Line Editor: (1) When designated as the first character of a string, requests lines beginning with the string, excluding the circumflex (for example, /^IDENTIFICATION/) and (2) Indicates negation in certain directives. Multi-User Debugger (Numeric) and $D Debug: Indicates negation as part of an IF directive.

_ (underscore)

File System: Joins two or more words in a file or directory name that the system is to interpret as one word (for example, LIST_PROG).

| (vertical bar)

Command Processor: Suppresses rescanning for returned active strings.

abbrev, login

See login abbreviation

abbreviation file

A file containing user-defined abbreviations and the character strings they represent.

abbreviation, login

See login abbreviation.

abbreviation processor

A system component that expands abbreviated commands and passes them to the Command Processor.

abort

An operator action resulting in the immediate cessation of operation of a task group or the operation of the currently executing request in a task group. All resources are returned to the Executive. The bound unit of the lead task of an aborted request may be retained.

absentee

A processing mode characterized by the absence of interaction between you and the system during execution of your program.

Access Control List (ACL)

A list specifying which user(s) can use the resource with which the list is associated.

ACL

See Access Control List.

activate

An operator action resulting in the resumption of a previously suspended task group. (See Suspend.)

active

A task is in the active state when it is executing or ready to execute, when its priority level becomes the highest active one in the central processor.

active function

A form of a command whose output string is placed in the command line before the rest of the line is processed.

active level

The priority level currently in effect.

address, absolute

A reference to a storage location that has a fixed displacement from absolute memory location zero.

address, relocatable

A reference to a storage location that has a fixed displacement from the program origin, but whose displacement from absolute memory location zero depends upon the loading address of the program.

administrator, system

Person responsible for registering users so that they can access the MOD 400 system.

after image

The image of a record in a restorable disk file as it exists after alteration. Written to a system journal file.

algorithm

A set of well defined rules for the solution of a problem.

alternate index organization

Alternate indexes are used to view a file ordered with a different key. The same data file can be ordered in many different ways by having more than one alternate index.

application program

A user-written program for the solution of a business, industrial, or scientific problem.

area

A DM6 I-D-S/II integrated file.

argument

User-selected items of data that are passed to a procedure (for example, system service macrocall arguments that are passed to the called system service, or command arguments passed to the invoked task). Synonymous with arg. (See parameter.)

argument, control

A keyword whose value specifies a command option. (See keyword.)

argument, positional

An argument whose position in the command line indicates to which variable the item of data is applied.

ASCII (American Standard Code for Information Interchange)

The interchange code established as standard by the American Standards Association.

asynchronous

Without regular time relationships. As applied to program execution, unpredictable with respect to time or instruction sequence.

attribute, file

Any of a set of disk file characteristics established when the file is created or modified to include such integrity features as recovery, restoration, and record locking.

base level

(See priority level, base.)

* BCB

(See Buffer Control Block.)

BCD

Binary-Coded Decimal notation.

before image

A copy of a record from a recoverable disk file, as it exists just prior to updating, written to a system recovery file.

Binary Synchronous Communications (BSC)

A communications procedure, using a standardized set of control characters and control character sequences, for the synchronous transmission of binary-coded data.

block

The logical unit of transfer between main memory and a tape file. The size of a block may be variable depending on the number of records and whether they are fixed or variable in length.

BMMU

(See MMU.)

bootstrap loader

(See loader, bootstrap.)

bootstrap routine

A routine, contained in a single record that is read into memory by a Read-Only Memory (ROM) bootstrap loader, which reads the operating system into memory. (See ROM bootstrap loader.)

bound unit

The output of one Linker execution that is placed in one file. A bound unit is an executable program consisting of a root and zero or more related overlays. The root and each overlay is also a load unit.

Bound Unit Descriptor (BUD) block

A system control structure containing information provided by the Linker to describe a bound unit.

break

A user action, initiated by pressing the break or interrupt key, that interrupts a running task so that commands can be entered. After the break, the interrupted task can be restarted or terminated.

breakpoint, bound unit

A point set in a debugging program where instructions are inserted to monitor the Executive loading process.

breakpoint, quick

A point in a program where a 02 instruction is inserted to monitor time-dependent tasks.

breakpoint, true

A point in a program where a 02 instruction is inserted to interrupt execution and activate a debugging program to monitor task execution.

broadcast

A message sent to all logged-on users through the Send Message Mailbox (SMM) command.

BSC

(See Binary Synchronous Communications.)

BUD

(See Bound Unit Descriptor (BUD) block).

Buffer Control Block (BCB)

A control structure, contained in the system pool area, which describes the characteristics of the buffer.

buffer, Input/Output (I/O)

A storage area used to compensate for the differences in the flow rates of data transmitted between peripheral devices and memory.

buffer pool

A collection of storage areas to which the File System assigns disk files when they are opened. Shared files are assigned to public pools in system memory. Exclusive files are assigned to private pools in task group memory (or to public pools if no private pools exist).

building, system

(See system building.)

bus

(See Megabus.)

byte

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

CACL

(See Common Access Control List.)

calling sequence

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

CCP

(See Channel Control Program.)

channel

A path along which communications can be sent.

Channel Control Program (CCP)

A microcoded program that resides in the Communications Processor; the CCP processes data characters, protocol headers, and framing characters.

checkpoint

A point in your program to which control can be returned and processing resumed following a task group abort. When you take a checkpoint, the system records the current contents of your memory area and the current status of tasks, files, and screen forms on a checkpoint file. (See restart.)

checkpoint file

A file on which the system records the current status of the task group request when a checkpoint is taken. Checkpoint files are created in pairs and checkpoints are written alternately to each file.

CI

(See Control Interval.)

CIP

(See Commercial Instruction Processor.)

cleanpoint

A point in your processing at which all file updates are considered to be valid. (See also rollback.)

CLM

(See Configuration Load Manager.)

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 timeslicing as well as requests to control tasks based on real-time considerations, and requests for the time of day and date in ASCII format.

Clock Request Block (CRB)

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.

code, object

The code produced by a compiler or the Assembler. The object code requires further processing by the Linker to produce a bound unit. (See also object unit.)

code, source

The code or language used by the programmer when the program was written. Code that must be processed by a compiler or the Assembler and the Linker before it can be executed.

cold restart

Restart after system failure.

command

An order that is processed by the Command Processor.

command accounting

A software component used to collect command-usage data and generate a report.

command-in

Any file or device from which commands to the Command Processor are read.

command language

The set of commands that you can issue to control the execution of your task.

command level

The state of the Command Processor, when it is capable of accepting commands, optionally indicated by the display of the RDY (ready) message.

command line

A string of up to 252 ASCII characters in the form: command_name_1 [arg_1...arg_n];command_name_2 [arg_1...arg_k] ..., where command_name_i is the pathname(s) of the bound unit(s) that performs the command's function. (See argument for a description of "arg."; see &.)

Command Processor

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

Commercial Instruction Processor (CIP)

A processor that includes an enhanced instruction set providing native commercial mode instructions.

commercial simulator

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

Common Access Control List (CACL)

A list specifying the access rights to all files or directories subordinate to the directory in which the list is established.

communications device

A device that transfers data over communications lines and is connected through a communications processor.

Compile Unit (CU)

A program unit, produced by a single execution of a compiler or the Assembler, that requires further processing by the Linker to produce a bound unit. (See object unit.)

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.

Configuration Load Manager (CLM)

A system component that reads a file of user-supplied directives and causes the system to be configured according to the contents of the directives.

control argument

(See argument, control.)

control character

An ASCII character intepreted by a device (such as a VIP) as having a keyboard control function.

Control Interval (CI)

The unit of transfer between main memory and the storage medium (primarily disk devices); comparable to a "block" for tapes. The size is specified by the user and remains constant for a file. For disk files, the size of the CI must be a multiple of 256 bytes. A Unified File Access System (UFAS) file is composed of CIs that are numbered, starting at one. The CI also determines the buffer size.

CRB

(See Clock Request Block.)

CRT

Cathode Ray Tube. (See Visual Information Projection.)

CTB

(See Clock Timer Block.)

CU

(See Compile Unit.)

cumulative file

A third disk file utilized by Level 4 error logging. (See hold file and raw file.) Statistics contained in the raw file can be analyzed by examining the contents of the cumulative file. The cumulative file contains performance histories for each monitored device or for memory. *

daemon

A system task group that manages queued print requests. |

Data Base Control Block (DBCB)

DM6 I-D-S/II working storage associated with a particular run unit containing record buffers, currencies, and other control information.

Data Base Control System (DBCS)

The DM6 I-D-S/II run-time package, which interprets Data Manipulation Language verbs, accesses the data base, and returns results to the user work area.

Data Description Language (DDL)

A nonprocedural language used to describe a data base (the schema description) or a portion of a data base (the subschema description).

data management

A File System component that handles the transfer of logical records.

DBCB

(See Data Base Control Block.)

DBCS

(See Data Base Control System.)

DDL

(See Data Description Language.)

device driver

A software component that controls all data transfers to or from a peripheral or communications device. (See line protocol handler.)

Device Media Control Language (DMCL)

A nonprocedural language that describes the physical characteristics of a DM6 I-D-S/II data base including CI size, area size, data base size, and CALC header frequency.

device-pac

The adapter between a Mass Storage Controller (MSC) or Multiple Device Controller (MDC) and peripheral device (for example, printer, diskette drive).

direct access

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

direct address

The method for reading or writing a record in any Unified File Access System (UFAS) file by supplying its simple key (control interval and line number).

directive

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

direct login

A login that Listener performs on a terminal as soon as that terminal comes online. The login is controlled by the terminal's T-record in the Terminals File. Compare manual login.

directory

A special file containing a description of other files and/or subordinate directories.

disk

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

disk cache processor

A separate processor with dedicated buffer memory that allows disk read requests to be processed at memory access speeds.

display processing

A method for developing, displaying, maintaining, and utilizing terminal display forms. (See VDAM and VISION.)

DM6 I-D-S/II

DM 6 Integrated-Data-Store/II. A CODASYL-based data base management system. (See also integrated file.)

DMCL

(See Device Media Control Language.)

dope vector

A structure for passing data items not aligned on word boundaries between programs.

dormant state

The state of a task when there is no current request for that task.

Dual-Line Communications Processor (DLCP)

A programmable interface between a central processor and communications device(s) consisting of two lines.

dynamic file organization

A file whose records are organized to be accessed sequentially or directly by their record position. The main purpose of this organization is to provide efficient storage for records to be accessed through alternate indexes.

EBCDIC

Extended Binary-Coded Decimal Interchange Code.

EC file

A file containing commands and (optionally) directives. In interactive mode, an EC file typically contains frequently used command sequences. In absentee mode, an EC file must contain all commands, directives, and anticipated user responses to program messages that will be needed for a session.

Edit Profile utility

An interactive program that allows the system administrator to register new users and/or to delete, list, enhance, and change the profiles of registered users.

EFN

(See External File Name.)

EMMU

(See MMU.)

entry point

A start address within the root segment of a bound unit. By default, an entry point is the beginning of a procedure; you can specify alternate entry point by symbolic address when you invoke 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 logging

The collecting of memory and/or hardware-related error statistics for selected peripheral devices.

error-out

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

*

extent

A group of contiguous allocated sectors on a disk.

External File Name (EFN)

The absolute pathname of any file within the system. It must start with the ^ or > character. It has the form ^vol_id>dir_1>...>dir_n>filename for files on logically dismountable volumes and the form (>)>dir_1>...>dir_n>filename for files on the system volume. Devices can be referred to by the sequence !sympd. (See sympd.)

external procedure

A routine that is assembled or compiled separately from the program that calls it.

FCB

(See File Control Block.)

FDB

(See File Description Block.)

FIB

(See File Information Block.)

field

A specific area of a record used for a particular category of data.

file

A named collection of one or more records.

File Control Block (FCB)

A File System data structure that controls a user's access to a file. An FCB is pointed to by an entry in the logical file table and, in turn, points to an FCB. There is one FCB per user Logical File Number (LFN) associated with a file.

File Description Block (FDB)

A File System data structure that describes a file or directory. An FDB is pointed to by an FCB for a particular file. There is one FDB per file or directory currently known (reserved) in the File System.

File Information Block (FIB)

A user-created data structure containing required information for file processing.

file management

A File System component that handles the creation, deletion, reservation, opening, and closing of files.

file name

A 1- to 12-character name assigned to a collection of related data records, or to a peripheral or communications device. For a file on disk, this name is assigned when the file is created. For devices, the name is assigned at system configuration. (See pathname.)

file organization

A method that establishes a relationship between a record and its location in a file. (See indexed, relative, random, dynamic disk, or sequential file organization.)

file recovery

Ability to bring an uncorrupted disk file to a consistent condition after a software malfunction or system failure.

file restoration

Ability to reconstruct a disk file that has been corrupted due to device fault.

file set

A number of tape volumes used to contain one or more files. There are four types of tape volumes:

1. Monofile Volume - Contains only one file

2. Multivolume File - Contains one file on two or more tape volumes

3. Multifile Volume - Contains more than one file on one volume

4. Multivolume Multifile - Contains more than one file with any file spanning more than one volume.

File System

System software consisting of file, data, and storage management components that handles Input/Output (I/O) functions of the supported I/O devices.

First-In/First-Out (FIFO)

An execution or scheduling algorithm in which the first item received is the first item processed.

fixed-length record

A record stored in a file in which all of the records are the same length.

floatable overlay

An overlay that can be loaded into any available memory location within a task group's memory pool.

form

A display terminal screen that provides areas (fields) into which you enter information that defines a function to be carried out.

full duplex

Simultaneous independent transmission of data in both directions.

full pathname

An absolute pathname which, when specified, begins with a circumflex (^) (for example, the root directory.)

function

A procedure that returns a single value to its caller. (See subroutine.)

globally sharable bound unit

A bound unit containing reentrant code and linked with the GSHARE directive. A globally sharable bound unit is loaded in the system pool and can be used by any task in the system.

group control block

A system structure describing attributes of a task group.

group_id

(See task group identification.)

group system space

An area of memory (segment) that contains the system control structures used to support a task group and its tasks in a swap pool.

group work space

An area of memory (segment) from which tasks in a swap pool obtain blocks of memory.

GRTS

General Remote Terminal Supervisor.

half duplex

Transmission of data in one direction at a time.

High Memory Address (HMA)

The address of the highest physical memory location in the central processor.

HMA

See High Memory Address.

hold file

A file that contains a copy of the Level 2 or Level 4 error logging statistics that are stored in memory. The hold file can be retrieved after system shutdown or crash.

*

home directory

Your initial working directory after logging in.

hot restart

Restart during a session.

I pool

(See independent memory pool.)

IMA

(See Immediate Memory Addressing.)

Immediate Memory Addressing (IMA)

A form of addressing a location in main memory by referencing the location directly, indirectly, or through direct or indirect indexing.

independent memory pool

A fixed partition memory pool. All tasks executing in a specific independent memory pool share a common virtual view consisting of all memory assigned to that pool and system global memory.

indexed file organization

A disk file whose records are organized to be accessed sequentially in key sequence or directly by key value.

indirect extent

The group of contiguous allocated disk sectors that holds the relative volume number that contains the succeeding set of extents.

Indirect Request Block (IRB)

(See Intermediate Request Block.)

Input/Output (I/O) device

A peripheral or communications device.

Input/Output Request Block (IORB)

A control structure used for communication between a program and an I/O driver outside of the File System.

integrated file

A data base disk file whose records are accessed directly or sequentially using CALC keys and key values.

interactive

A processing mode characterized by a dialog between you and the system during execution of your program.

interactive task

A task, which, when invoked, is under real-time control of user-specified directives.

Intermediate Request Block

A control structure used by the system to queue and record status and control information developed from user request blocks; also known as Indirect Request Blocks.

international graphics

ASCII characters whose hexadecimal equivalents are C0 through FF.

interrupt

The initiation, by hardware, of a routine intended to respond to an external (device-originated) or internal (software-originated) event that is either unrelated, or asynchronous with, the executing program.

Interrupt Save Area (ISA)

An area used to store the context of an interrupted task. There is one ISA for each task in memory.

interrupt vector

A pointer to a priority-level-specific memory area called an ISA. There is one vector for each priority level, each having a dedicated memory location.

Intersystem Link (ISL)

A hardware element interconnecting two buses, thereby permitting the same functions between two units on different buses as between two units on the same bus.

IORB

(See Input/Output Request Block.)

ISA

(See Interrupt Save Area.)

ISL

(See Intersystem Link.)

journal file

A system file that contains a running summary of all changes made to all disk files designated as restorable.

key

An identifier for a specific record within a disk file.

keyword

A fixed-form character string preceded by a hyphen (for example, -ECL). It can stand alone (for example, -WAIT) or can be followed by a value (for example, -FROM n).

KSR

A Keyboard Send-Receive teleprinter.

KSR-like terminal

A KSR teleprinter, CRT keyboard, or Visual Information Projection (VIP) terminal, which supports the Teleprinter (TTY*) protocol and is connected to the MDC, MLCP, or DLCP.

language key

A two-ASCII-character identifier used as a file name suffix to provide multiple national language support. The system default language key is specified at CLM time with the system default message library pathname. *

lead task

The controlling task of a task group. The lead task can invoke other tasks to perform functions on its behalf (for example, system services).

LFN

(See Logical File Number.)

LFT

(See Logical File Table.)

line

A record stored in a Series 60-compatible file.

line number

The relative position of a logical record within a control interval (CI). Line numbers start at zero for each CI.

Line Protocol Handler (LPH)

A communications program that processes messages, interrupts, and timeouts; handles protocol acknowledgment and error recovery; initializes the channel control program.

link

(1) A process by which the Linker program combines separately compiled object units to produce a bound unit. (2) A communications channel between two modems. (3) A name that refers to a File System entity. A link establishes an additional pathname to a file, directory, or index in another volume or directory so that it can be referenced as if it were contained in the specified directory.

Linker

A utility program that links one or more object units into a single machine language relocatable unit. |

Listener

A system control component that allows you to access the system through a selected set of terminals by means of Login commands.

load unit

A discrete program unit that has been compiled or assembled and linked. It is in machine language and is directly loadable by the Executive. See bound unit.

Loader

A system control software component that dynamically loads from disk the root and overlays of a bound unit.

Loader, bootstrap

A utility program, usually permanently resident in main memory, that enables other programs to load themselves.

Logical File Number (LFN)

An internal identifier that becomes associated with a file when it is reserved. LFNs are used in all file references until the file is removed.

Logical File Table (LFT)

A data structure for use by the File System. It contains an entry for each LFT.

Logical Resource Number (LRN)

An internal identifier used to refer to tasks or devices.

Logical Resource Table (LRT)

A data structure within a task group containing an entry for each LRN used in an application, or a data structure within a system task group containing an entry for each LRN representing a device. Each entry is a pointer to the Resource Control Table (RCT).

Login

A command entered at a terminal monitored by Listener that is used to gain access to the system. The Login command spawns a task group to be associated with your terminal for a primary login or passes you to an existing task group for a secondary login.

login abbreviation

A one-character type-in that is defined in the terminals file as an abbreviation for a complete login line. A login abbreviation may apply only to a specific terminal or may be used at all terminals in the system.

login identification (login id)

A string that identifies a registered user of the system for purposes of login only. It contains no periods (.); uppercase and lowercase characters are distinct (JONES and Jones are different login ids).

login parameters, default

Login line parameters stored in your user profile. When you log in, these parameters are combined with arguments from the terminals file and/or arguments entered manually at login time to form the actual login line.

LPH

(See Line Protocol Handler.)

LRN

(See Logical Resource Number.)

LRT

(See Logical Resource Table.)

mail

Data that is accompanied by routing information and is contained in a mailbox.

mailbox

A special file that may contain data to be communicated to another task group (user).

manual login

The procedure by which you enter a login line or fill out a login form in order to log in at a terminal. Compare direct login.

MBZ

Must Be Zero.

MDC

Multiple Device Controller for peripheral devices other than cartridge disk, storage module, and magnetic tape.

Megabus

A set of parallel conducting paths connecting various hardware units of a computer.

memory dump

The representation of the contents of memory.

Memory Management Unit (MMU)

A hardware feature that intercepts all addresses generated by the Central Processor Unit (virtual addresses) and transforms them to real memory addresses via its mapping array. There are two types of memory management units, the basic memory management unit (BMMU) and the extended memory management unit (EMMU).

Memory Manager

A system control software component that controls dynamic requests to obtain/return memory from/to a memory pool.

memory pool

A block of central processor memory from which a task group obtains memory as required for executable code, control structures, and I/O buffers. (See swap, independent, or system pool.)

memory save and autorestart unit

A hardware feature that can preserve the memory image through a power failure lasting up to 2 hours.

menu

A display on a terminal screen that lists two or more selections from which you can make a choice.

Menu Processor

A software component that interprets commands issued through the User Productivity Facility, and invokes the required function.

menu subsystem

A system component known as the User Productivity Facility that enables you to communicate with the Executive through menus and forms.

message

A communication of text that is to be displayed immediately at the receiving user's terminal.

message facility

A system component that provides a means for intertask and intergroup communications.

message reporter

A system component that extracts messages from the message library, formats them, and delivers them to a user task. *

MLCP

(See Multiline Communications Processor.)

MMU

(See Memory Management Unit.)

MSC

Mass Storage Controller for cartridge disks or storage modules.

MTC

Magnetic Tape Controller for magnetic tapes.

Multiline Communications Processor (MLCP)

A programmable interface between a central processor and one or more communications devices. Can be programmed to handle specific communications devices.

multiprogramming

An operating system capability that allows the concurrent execution of tasks from more than one task group.

multitasking

An operating system capability that allows the concurrent execution of more than one task in one or more task groups.

multivolume set

A number of disk volumes that contain one or more files. An online multivolume set allows data for a single file to be distributed over many volumes. It requires that all volumes be mounted and available for the file to be used. A serial multivolume set permits sequential files to extend onto other volumes. The volumes can be mounted one at a time and can be used for very large sequential files.

NATSAP

Next Available Trap Save Area Pointer.

* nonfloatable overlay

An overlay that is loaded into the same memory location relative to the root each time that it is loaded.

* OAT

(See Overlay Area Table.)

object unit

A relocatable program unit produced by a single execution of a language compiler, or by the Assembler, and requiring further processing by the Linker to produce a bound unit.

OIM

(See Operator Interface Manager.)

OIM log

A system facility that is used to capture all traffic on the operator's console.

online

An execution environment intended for use by application programs, including those operating in real time.

* online task group

A task group that executes in the online dimension; its resources are a memory pool and the peripheral devices it requests.

operating system area

The memory area containing operating system software, user-written extensions to the operating system, and device drivers.

operator

Person who starts up the system each day, controls processing, manages peripheral devices, monitors system states, and regulates absentee jobs.

operator commands

The set of commands that can be issued by the operator to control execution.

Operator Interface Manager (OIM)

A system control software component that manages all messages sent simultaneously by multiple task groups to the operator terminal or from the operator terminal to a task group.

operator-out

The file or device by which an interactive command communicates with the system operator; established at system initialization or when a File Out command is issued.

operator terminal

A terminal specified for use in interactive communications between the operator and vendor-supplied and user-written application programs.

overlay

A section of a program that can be loaded during execution to overlay another section of the program. Used when there is insufficient memory to accommodate all the code of a program. (See floatable overlay and nonfloatable overlay.)

overlay area

An area of specified size into which floatable overlays are loaded.

Overlay Area Table (OAT)

A data structure containing parameters that control the use of overlay areas.

pacing rate

The frequency at which each new output line appears on an output display.

parameter

The data received by a procedure that is written in a generalized form to handle any data passed to it. See argument.

password

A unique combination of characters that identifies you to the system. A system control component verifies the password before granting access to the system.

patch

A portion of code used to modify an existing object or load unit on disk or in memory.

pathname

A character string by which a file, directory, or device is known in the File System.

pathname, absolute

A pathname that begins with a greater-than sign (>) or a circumflex (^). In the former case, it is a partial pathname and is appended to the root directory name of the system volume to form a full pathname; in the latter case, it is a full pathname and is used without modification.

pathname, device

A pathname by which reference is made to a peripheral device. Device pathnames have the general form !device_id.

pathname, relative

A pathname that does not begin with a greater-than sign (>) or a circumflex (^). It is a partial pathname consisting of one or more directory names and/or a file name, and is appended to the working directory pathname to form a full pathname.

pathname, simple

A special form of a relative pathname consisting of a single directory name or file name. It is appended to the working directory name to form the full pathname.

peripheral device

A device connected through the MDC, MSC, or MTC (for example, a card reader, disk, or tape).

Physical Input/Output (PIO)

Physical Input/Output, or physical I/O, that is initiated through a request I/O macrocall, outside of the File System, using IORBs.

PIO

(See Physical Input/Output.)

pipe

A special kind of sequential file used for synchronizing and passing information among multiple cooperating tasks.

pool identifier

A two-character name, established a system configuration, by which a memory pool is identified, and by which a task group is assigned a memory pool when the task group is created.

positional argument

(See argument, positional.)

power resumption

A system facility that controls the restarting of the execution environment following a power failure.

primary login

The form of login that requests Listener to spawn a task group that has the terminal from which the login originated as its primary system file (the terminal will be the initial user-in, command-in, error-out, and user-out files).

priority level

A numeric value that can be assigned to a task or device for purposes of controlling processing. Values range from 0 to 63. The lowest values (highest priorities) are reserved for system tasks; level 63 is the system idle level. Intermediate levels are available for user assignment to tasks and devices. The physical level at which a task executes is the sum of the highest level number assigned to a configured device plus three, the base level of the task group, and the relative level of the task within the group.

priority level, base

The priority level, relative to the system priority level, at which all tasks in a task group execute. A base level of 0 is the next higher level above the last (highest) system priority level.

priority level, hardware

A numeric value from 0 through 63 that can be assigned to a task or device to control processing. The lowest values (highest priorities) are reserved for certain system tasks. Level 62 is reserved for user tasks. Level 63 is the system halt level.

priority level, physical

(See priority level.)

priority level, relative

The priority level, relative to the base level, at which a user task within a task group executes. Relative Level 0 is the base level.

priority level, system

The priority level assigned to system devices and tasks.

profile

(See report queue profile file or user profile.)

program name suffixes

A "point-letter" character string such as ".O" for object units or ".A" for Assembly language source units appended to a file name to identify it as a source, object, or list unit.

protected string

A character string containing reserved characters that is enclosed by protected string designators. (See reserved character and protected string designator.)

protected string designator

A pair of quotation marks or apostrophes that enclose a character string containing reserved characters. (See reserved character.)

PVE

Polled Visual Information Projection (VIP) Emulator.

quarantine unit

A unit of message text; the smallest amount of transmitted data that is available to the receiver.

query

A collection of command statements that causes a query processor to examine a file (or data base) and produce a written report.

random file organization

A disk file whose records are accessed directly or sequentially through CALC keys and key values.

range

The number of bytes transferred during an I/O operation.

raw file

A second disk file utilized by Level 4 error logging. The raw file maintains a cumulative performance record for memory and/or each device being monitored.

record

A user-created collection of logically related data fields. Records are treated as units and can be fixed or variable in length.

record locking

A file access feature that controls contention for records within disk files shared by two or more task groups.

recoverable file

A disk file that has been identified as one that can be brought back to a previously established state in the event of a software malfunction or system failure. (See before image and file recovery.)

recovery file

A system-created file used to contain before images. (See before image.)

reentrant routine

A routine that does not alter itself during execution; a reentrant routine can be entered and reused at any time by any number of callers.

registration

Process by which the system administrator introduces users into the system.

relative file organization

A file whose records are organized to be accessed sequentially or directly by their record position relative to the beginning of the file.

relative level

(See priority level, relative.)

relative record number

A number representing the position of a record relative to the beginning of the file. The initial record is relative record number 1.

remote file access

A File System facility that allows applications to access remote data as if it were local.

report queue

A directory used to contain the pathnames of files queued for later transcription.

report queue profile file

A file that designates the characteristics of reports that will be entered in a report queue and printed or punched at a later time.

report spooling

The queuing and subsequent transcription of reports.

request block

(See Input/Output Request Block (IORB), Task Request Block (TRB), Clock Request Block (CRB), and Semaphore Request Block (SRB).)

request I/O

The macrocall, issued to a driver, that performs Physical I/O (PIO).

request queue

A threaded list of request blocks.

reserved character

An ASCII character to which special significance is attached. These characters are: space (blank), horizontal tab, quotation mark ("), apostrophe ('), semicolon (;), ampersand (&), vertical bar (|), left bracket ([), and right bracket (]).

resident bound unit

A bound unit that is permanently configured in memory as an extension to the operating system.

residual range

The difference between the number of bytes requested and the number of bytes transferred during an I/O operation.

Resource Control Table (RCT)

A system structure that controls task processing.

restart

A user-initiated process in which the system locates the most recently completed checkpoint on the checkpoint file and reads the checkpoint image, rebuilding the Executive data structures and memory blocks, reloading bound units, and repositioning active user files. (See checkpoint.)

restorable file

A disk file that has been identified as one that can be reconstructed to its latest state following a device fault. (See after image and file restoration.)

return address

The address of the instruction in a program to which control is returned after a call to a subroutine. By convention, this address is usually stored in register B5.

RFU

Reserved for Future Use.

ring

A mechanism used to establish access rights to a segment. Also an attribute of a memory pool.

rollback

The process by which before images stored on a recovery file are written to a recoverable file, negating updates made since the last cleanpoint. This action restores the file to the state it was in when the cleanpoint was taken. Also see cleanpoint, before image, file recovery, and recoverable file.

ROM bootstrap loader

A firmware routine (activated by pushing the Load key on the control panel) that reads the first record from a designated disk into memory.

root directory

The primary directory on a mass storage volume; it is pointed to by the root directory pointer in the volume label. The name of the root directory is the same as the vol_id. MOD 400 supports a User Root Directory (URD) and a System Root Directory (SRD), which may reside on different volumes.

root segment

The controlling segment of a program. It is resident in memory during the entire execution of the program and can call overlay segments.

ROP

Receive-Only Printer.

RSU

Reserved for System Use.

Scientific Instruction Processor (SIP)

A processor that executes a set of scientific instructions.

search rules

An ordered list of directories searched by the system when a bound unit is to be located and loaded or executed.

secondary login

The form of login that requests the Listener to transfer control of your terminal to a specified task group. The specified task group must already exist and have an outstanding Request Terminal monitor call ($RQTML) for the secondary login to satisfy.

secondary user

A user whose login line contains a destination, which is the identification (usually by group-id) of a subsystem that has requested a secondary terminal. The user is attached to the subsystem until released by it.

sector

A 128-byte portion of a diskette track, or a 256-byte portion of a diskette, cartridge disk, cartridge module disk, or storage module track.

security

Limitation and control of the type of access a user has to directories, files, and the system itself.

segment

A logical entity containing a bound unit root and zero or more additional load elements, or containing one or more overlays.

semaphore

A software counter mechanism, available to Assembly language programs, and used to coordinate the use of task code or other resources such as files.

Semaphore Request Block (SRB)

A data structure used to control semaphore processing.

sequence number

The internal identification number assigned to a request in a task group request queue.

sequential access

The method of reading or writing a record in a file by requesting the next record in sequence.

sequential file organization

A file on disk or magnetic tape whose records are organized to be accessed in consecutive order.

sharable bound unit

A transient bound unit consisting of reentrant code linked with the share directive. A sharable bound unit is available for execution by any task assigned to the same memory pool.

sharable file

Any file that is usable by more than one task concurrently.

SIP

(See Scientific Instruction Processor.)

SIP Simulator

A software component that provides the same functionality as the SIP.

source unit

A program written in source language for processing by a compiler or an assembler. Source units are stored as variable sequential data files.

spanned record

A record that spans a control interval or block.

spawn

To create, request the execution of, and then delete a task or task group.

spooling

The technique for storing output on disk files for subsequent printing or punching.

SRB

(See Semaphore Request Block.)

standard I/O files

The command-in, user-in, user-out, operator-out, and error-out files.

star name convention

A special pathname convention that can be used with certain commands to perform an operation on a group of files, thereby eliminating the need for separate commands for each file.

startup

The procedure that bootstraps a vendor-supplied, preconfigured system from disk to provide a minimum operating environment.

startup EC file

An EC file whose commands are executed at system startup or when a task group is activated.

states (task)

A task can be in the following states: Dormant, Active, Wait, and Suspend.

storage management

The File System component that handles the transfer of blocks and control intervals between main memory and secondary storage (for example, disks, tapes, etc.).

subroutine

Any procedure that alters data in an area common to both the subroutine and its caller. Contrast with "function".

subsystem

A general purpose application-oriented facility that provides interactive users with their interactive capabilities and view of the system. A subsystem is generally identified directly with the lead task of a task group. A subsystem can either be primary-user oriented (supporting one interactive user per task group) or secondary-user oriented (supporting multiple interactive secondary users per task group).

subsystem switcher

A menu-oriented component of the user productivity facility that allows a logged in user to switch from one subsystem to another without having to log out and back in again.

suspend

An operator action resulting in the temporary cessation of execution of a task group; all resources are retained by the task group. (See activate.)

swap pool

A memory pool in which segmented memory management is used. Tasks assigned to a swap pool can be swapped to backing storage in order to make memory available to competing tasks. Each task executing in a swap pool has its own virtual view. The system allows multiple swap pools to be assigned.

Swapper

A system component that controls the allocation of swap pool memory and swap file space.

symbolic start address

Bound unit entry point.

sympd

A name assigned to each peripheral device when the system is built. The acronym sympd stands for "symbolic peripheral device."

system building

The process of specifying system variables, identifying the peripheral devices and (optional) communications environment to the system, and tailoring main memory to suit system and user needs.

system console

(See operator terminal.)

system directory

One of the directories that the system uses in its search for a bound unit to be loaded for execution.

system global space

The memory of the fixed system area and the system. This memory is in the virtual view of all tasks, regardless of their specific memory pool assignments.

system pool

The memory area from which the system task group (GCB and TCB) and system global structures (for example, BCB and FDB) are allocated, and the area where globally sharable bound units reside.

system service macrocalls

Macrocalls available to Assembly language programs to perform a wide variety of system control and File System service functions.

system task group

The task group in which all drivers, the clock, the Command Processor, and OIM execute.

task

A sequence of instructions that has a starting point, an ending point, and performs some identifiable function.

task address space

The boundaries of a task in a swap pool. Consists of bound units, user stack segment, dynamically created segments, group work space, group system space, and system global space.

Task Control Block (TCB)

The system control structure that describes the task's characteristics, including the contents of the hardware Interrupt Save Area (ISA).

task group

A named set of one or more tasks with a common set of resources; the framework within which every user and system function operates.

task group identification

A two-character name by which a task group is known to the system.

task group resource

One of a set of elements associated with a task group that enables it to perform its function. A resource can be a task, a central processor priority level, central processor memory, or a peripheral or communications device.

Task Manager

A system component that handles task requests to activate, wait, or terminate tasks.

Task Request Block (TRB)

A data structure used by one task to request another task and communicate with it.

TCB

(See Task Control Block.)

terminal

An I/O device.

terminals file

A sequential file that names the terminals monitored by Listener, defines terminal-specific access constraints, and defines system-wide and terminal-specific abbreviations for login lines.

terminate

A system service macrocall request issued by the currently executing task at the end of its normal processing.

terminated

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

timeslicing

An optional feature that minimizes the ability of tasks that use large amounts of central processor time to interfere with interactive users.

transaction

An event that is entered, recorded, and processed by the system.

transaction processing

Online data processing in which individual transactions are entered from terminals, validated, and processed through all relevant procedures.

transient bound unit

A bound unit that resides in memory as long as there is a request for it.

transparent mode transmission

A data transmision mode that allows data consisting of bytes having any bit configuration to be transmitted over communications lines. Thus, control characters can be transmitted as data.

trap

A control transfer caused by an executing program. The transfer is made to a predefined location in response to an event that occurs during processing.

trap handler

A routine designed to take a particular action in response to a specific trap condition.

Trap Manager

A system control software component that handles an executing program's transfer of execution control to a predefined trap location.

Trap Save Area (TSA)

An area in memory in which certain information is stored when a trap occurs.

trap vector

A pointer to a trap handler. There is one vector for each possible trap condition, in dedicated memory locations.

TRB

(See Task Request Block.)

TSA

(See Trap Save Area.)

UFAS

(See Unified File Access System.)

Unified File Access System (UFAS)

A file organization developed to provide a predictable relationship between records and their location in the file. UFAS files are transportable across all levels of Series 60 software.

unit control character

(See control character.)

user

An entity that can make demands upon the system; can be a logged-in person, a system routine such as a daemon, etc. A person logged in under two accounts is considered to be two users for system loading purposes.

user identification (user id)

A string that identifies the current user of a task group. It consists of two or three parts: person.account[.mode]. Uppercase and lowercase letters are treated the same (JONES.ADMIN and Jones.Admin are treated the same).

user-in

The file or device from which a command function requiring directives (for example, the Line Editor) reads its input; it is established when the group request is made. User programs can also read from this file.

user-out

The file or device by which an interactive command communicates with the user; established when a group request is made, or a File Out command is issued. User programs can also write to this file.

user productivity facility

A MOD 400 interface that allows you to communicate with the Executive through menus and forms instead of commands. You select a function from a menu and fill in the resulting form to accomplish a specific function.

user profile

The user's registration information as maintained by the system administrator using the Edit Profile utility. The user profile establishes a login id and a unique password capability for each user, as well as other privileges and/or limitations granted to specific users.

user registration

A mode of MOD 400 operation that maintains a file of registered users that specifies their login defaults and individual access rights. For definitions of terms related to user registration, see the Glossary in the System Building and Administration manual.

user stack segment

A work area available through hardware stack instructions.

variable-length record

A record stored in a file in which records have different lengths.

VDAM

A system component that allows you to display and use forms.

VIP

(See Visual Information Projection.)

virtual view

A virtual view consists of all of the memory pools to which a task executing within the view has access. A virtual view consists of one of the following combinations of memory pools:

- The system pool and an independent pool.
- The system pool and a swap pool.

VISION

A system component that allows you to develop and maintain forms.

Visual Information Projection (VIP)

VIP devices consist of a screen (CRT) and keyboard. Hard-copy receive-only printers can be added to some models.

vol_id

(See volume identifier.)

volume

A fixed or removable storage unit (for example, storage modules, diskettes, cartridges, tapes) that may contain one or more files.

volume header

A unique record at the beginning of every disk or magnetic tape volume that carries information about the volume.

volume identifier (vol_id)

The unique name for a disk or magnetic tape volume that is contained in the volume header.

volume name

(See root directory.)

volume set

A number of disk volumes that contain one or more files. Online volume sets require that all volumes are mounted and are available for use. Serial volume sets can be mounted one volume at a time.

wait

A task is in the wait state when it causes its own execution to be interrupted until a time request is satisfied, until another task releases a semaphore, until another task terminates, or until an I/O operation terminates.

word

A sequence of 16 consecutive binary digits operated upon as a unit; two consecutive bytes.

working directory

A disk directory pathname associated with a task group. It begins with a root directory name and contains zero or more intermediate directory names. It is used by the File System software to construct a full pathname whenever a task group refers to a relative or simple pathname.

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	DPS 6 GCOS 6 MOD 400 SYSTEM CONCEPTS	ORDER NO.	CZ03-01
		DATED	MARCH 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: DPS 6
GCOS 6 MOD 400
SYSTEM CONCEPTS

ORDER NO. CZ03-01

DATED MARCH 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: DPS 6
GCOS 6 MOD 400
SYSTEM CONCEPTS

ORDER NO. CZ03-01

DATED MARCH 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Kanda Jimbo-cho Chiyoda-ku, Tokyo
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

44347, 186, Printed in U.S.A.

CZ03-01

DPS 6
GCOS 6 MOD 400
SYSTEM CONCEPTS
ADDENDUM A

SUBJECT

Additions and Changes to the Manual

SPECIAL INSTRUCTIONS

This is the first addendum to CZ03-01, dated March 1986. Insert the attached pages into the manual according to the collating instructions on the back of this sheet. Change bars in the margin indicate new or changed information; asterisks indicate deletions.

Note:

Insert this cover sheet behind the front cover to indicate the updating of the document with Addendum A.

SOFTWARE SUPPORTED

This document supports Release 4.0 of the MOD 400 Executive.

ORDER NUMBER

CZ03-01A

September 1986

45943
0886
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

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

Remove	Insert
iii through xiv	iii, blank v through x xi, blank
xv, blank	
1-5, blank	1-5, blank
2-5, 2-6	2-5, 2-6
2-35 through 2-40	2-35 through 2-40
3-1, 3-2	3-1, 3-2
	3-2.1, blank
3-17, 3-18	3-17, 3-18
3-19, blank	3-19, blank
4-7, 4-8	4-7, 4-8
5-1, 5-2	5-1, 5-2
5-5 through 5-10	5-5 through 5-10
5-23, 5-24	5-23, 5-24
6-3, 6-4	6-3, 6-4
g-1 through g-8	g-1 through g-8
g-21, g-22	g-21, g-22
g-33, g-34	g-33, g-34
	h-1, blank

USER COMMENTS FORMS are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

File No.: 1R13

09/86
CZ03-01A

- Backup and recovery facilities, including the backup and restoration of disk files, the preservation of the execution environment during a power failure, the recovery of files at the record level, and the recovery and restart of task groups.

Although no manual is prerequisite to this manual, you may find it convenient to have read the Guide to Software Documentation.

Each section/appendix of this document is structured according to the heading hierarchy shown below. Each heading indicates the relative level of the text that follows it.

Level	Heading Format
1 (highest)	ALL CAPITAL LETTERS, UNDERLINED
2	Initial Capital Letters, underlined
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, not underlined

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

ILLUSTRATIONS

TABLES

The Distributed Systems Architecture 6 (DSA6) package follows the layered structure of the Open Systems Interconnection (OSI) defined by the International Standards Organization. DSA6 is a set of networking products that includes a transport facility, a network terminal manager, a unified file transfer facility, a remote file facility, a remote batch facility, and an application interface facility. DSA6 also supports system to system Local Area Network (LAN) connections. In addition, DSA6 provides terminal access to IBM-hosted applications through the DSA/SNA gateway.

The Systems Network Architecture 6 (SNA6) package emulates most operations of standard IBM devices so that DPS 6 systems can interface with an IBM SNA network. SNA6 provides a remote job entry facility, a file transmission facility, an interactive terminal facility, and an application interface facility.

Data Management Software

MOD 400 supports data base management, query and report writing, and transaction processing software packages. Data base management packages are available for relational and network data bases. Query and report writing packages allow you to retrieve information from all supported data bases. Transaction processing packages support standalone systems as well as applications connecting to remote host processors through the Distributed Systems software.

Data Entry Software

MOD 400 supports a multistation, forms-oriented source data collection capability. The Data Entry Facility-II (DEF-II) package embodies established data entry concepts in a menu-driven approach, making it easy to specialize and run procedures. Data collected and validated by DEF-II can be organized into a file and transferred to another system through the Distributed Systems software.

Office Automation Software

MOD 400 supports the Office Automation System (OAS) facility. OAS offers a wide range of office processing functions including document processing, electronic mail, document transfer, records processing, spreadsheets, communications, and file management.

WORKING DIRECTORY

The File System always starts at a root directory when it searches for a disk file or a directory. At times the search for an element residing on a disk volume may traverse a number of intermediate directory levels before the desired element is located, and the File System must be supplied with the names of all the directories it must pass on the way. Frequently all files of interest to a user doing work on the system are contained in a single directory that is three or four levels deep in the hierarchy. It is convenient to be able to refer to files in relation to a directory at some arbitrary level in the hierarchy rather than in relation to the root directory. The File System allows this to be done by recognizing a special kind of directory known as a working directory.

A working directory establishes a reference point that enables you to specify the name of a file or another directory in terms of its position relative to the working directory. If the access path of the working directory is made known to the File System, and if the desired element is contained in that directory, the element can be specified by just its name. The File System concatenates this name with the names of the elements of the working directory's access path to form the complete access path to the element.

Disk Directory and File Locations

The File System has total control over the physical location of space allocated to directories and files. You need never be concerned about where a directory or file resides on a volume. When a volume is first initialized, space is allocated to elements in essentially the order in which they are created. But, after the volume has been in use for some time, elements may have been deleted and the space they occupied made reusable. Then, when a new element is created, it is allocated the first available space. If more space is needed, it is obtained from the next free area.

Disk Directory and File Naming Conventions

Each disk directory and file name in the File System can consist of the following American Standard Code for Information Interchange (ASCII) characters:

- Uppercase and lowercase primary character set alphabetics (A-Z, a-z)
- Digits (0-9)
- Underscore (_)
- Hyphen (-)

- Period (.)
- Apostrophe (')
- Uppercase and lowercase characters whose hexadecimal equivalents are from C0-FE (Western European Latin alphabet, also called the extended character set).

The characters in the extended character set cannot be used in volume identifiers.

NOTE

If the terminal is not capable of processing 8-bit data, characters from the extended character set are displayed as periods or as their 7-bit equivalents.

When volumes, files, and directories are created, their identifiers are stored on disk exactly as entered, in uppercase and lowercase characters. For both the primary and extended character sets, MOD 400 considers uppercase and lowercase characters to be equivalent (for example, "DATA", "Data", and "data" all refer to the same file).

The first character of any name must not be the character FF (hexadecimal). The underscore character can be used to join two or more words that are to be interpreted as a single name (for example, DATE_TIME). The period character and one or more following alphabetic or numeric characters are normally interpreted as a suffix to a file name. This convention is followed, for example, by a compiler when it generates a file that is to be listed. The compiler identifies this file by creating a name of the form FILEA.L.

MAXIMUM NAME LENGTH

The name of a root directory (the volume identifier) can be from one through six characters in length. The names of other directories and files can be from 1 through 12 characters in length. The length of a file name must be such that any system-supplied suffix does not result in a name containing more than 12 characters.

UNIQUENESS OF NAMES

Within the system at any given time, the access path to every element must be unique. This requirement leads to the following rules for naming files:

If the system volume is associated with the disk cache processor, heavily used directories, as well as files that are read sequentially, are likely to be resident in the disk cache buffer. Buffer pools for these directories and files may not be needed.

- Private buffer pools - Private buffer pools can be created by each user. Private buffer pools reside in the task group's memory space and are available only for disk files reserved exclusively by that task group. A disk file is assigned to a private pool if the file is reserved for exclusive use and its control interval size (specified in the command that creates the file) matches the pool's buffer size. Private buffer pools should be created only if necessary to meet specific buffering needs. Public buffer pools should be sufficient in most cases.

- File-specific buffer pools - When you reserve a disk file with the Get File command, you can specify the number of buffers (using the -NBF argument) to be used when accessing the file. When the file is opened, a buffer pool is automatically created for use only by that file. This file specific pool is created in the task group's memory if the file is reserved exclusively, or in system memory if the file is reserved as sharable. The -NBF argument should be used carefully since it prevents a file from being assigned to a public or private buffer pool.

Buffer Pool Optimization

The File System collects a set of statistics on the use of each buffer pool. The installation can use this information to optimize disk I/O operations. Buffer pool statistics are obtained through the Buffer Pool Status and Buffer Pool Information commands. The Buffer Pool Status command provides a summary of the public or private buffer pool status. The Buffer Pool Information command provides a detailed status report on a particular buffer pool.

The installation should analyze applications and their associated file usage to fully utilize the advantages offered by buffer pools. Only a limited number of control interval sizes should be allowed for user files. In general, buffer and control interval sizes should be chosen to evenly distribute high and low activity files over the various buffer pools, thus reducing the amount of contention in the pools. The initial determinations will provide an acceptable level of performance and provide the basis for further analysis.

The Adjust Buffer Pool command can be used to temporarily alter the number of buffers in a private buffer pool. Once the most efficient buffer pool size has been established, it should be permanently fixed through the Create Buffer Pool and Delete Buffer Pool commands.

MAGNETIC TAPE FILE CONVENTIONS

The magnetic tape file conventions discussed in the following paragraphs include file organization, naming conventions, pathnames, and buffering operations.

Tape File Organization

The following information applies only to 1/2-inch, 9-track magnetic tapes.

Magnetic tape supports only the sequential file organization. Fixed- or variable-length records can 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.

The unit of transfer between memory and a tape file is a block. Block size varies depending on the number of records and whether the records are fixed or variable in length.

A block can be treated as one logical record called an "undefined" record. An undefined record is read or written without being blocked, unblocked, or otherwise altered by data management. Spanned records (those that span across two or more blocks) are supported. No record positioning is allowed with spanned records.

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

- Single-volume, single-file
- Multivolume, single-file
- Single-volume, multifile
- Multivolume, multifile.

The following types of unlabeled tapes are supported:

- Single-volume, single-file
- Single-volume, multifile.

Magnetic Tape File and Volume Names

Each tape file and volume name in the File System can consist of the following ASCII characters: Uppercase alphabetics (A through Z), lowercase alphabetics (a through z), digits (0 through 9), exclamation point (!), double quotation marks ("), dollar sign ($), percent sign (%), ampersand (&), apostrophe ('), left parenthesis ((), right parenthesis ()), asterisk (*), plus sign (+), comma (,), hyphen (-), period (.), slash (/), colon (:), semicolon (;), less-than sign (<), equal sign (=), question * mark (?), and underscore (_).

The underscore character can be used as a substitute for a space. If a lowercase alphabetic character is used, it is converted to its uppercase counterpart ("DATA", "Data" and "data" all refer to the same file). *

Any of the characters defined above can be used as the first character of a file or volume name.

The name of a tape volume can be from 1 through 6 characters in length. Tape file names can be from 1 through 17 characters.

Magnetic Tape Device Pathname Construction

Magnetic tape volumes can be labeled or unlabeled (refer to "Tape File Organizations" above).

UNLABELED TAPE PATHNAME

You must use a device pathname when referring to an unlabeled tape. The general form of a tape device file pathname is:

```
!dev_name
```

where dev_name is the symbolic name defined for the tape device at system building time.

LABELED TAPE PATHNAME

You can refer to labeled tapes either by the tape device file pathname convention or by the tape volume id convention.

The tape device file pathname convention is:

```
!dev_name>vol_id[>filename]
```

where dev_name is the name of the tape device as specified at system building time, vol_id is the name of the tape volume, and filename is the name of the file on the volume. This convention requires that the volume be mounted on the specified device.

The tape volume id convention is:

```
~vol_id[>filename]
```

where vol_id is the name of the tape volume and filename is the name of the file on the volume. This convention allows the volume to be mounted on any available tape device.

Automatic Tape Volume Recognition

Automatic volume recognition dynamically notes the mounting of a tape volume. This feature allows the File System to record the volume identification in a device table, thus making every tape volume accessible to the File System software.

Magnetic Tape Buffering

The -NBF argument of the Get File command can be used with magnetic tape files to reserve one or two buffers. If -NBF is not used, the File System attempts to allocate two buffers. If two buffers are allocated, the File System does "double buffering." When the tape file is being read, the File System unblocks one buffer while an anticipatory read is done into the other buffer. Similarly, when the tape file is being written, the File System blocks records into one buffer while a previously filled block is written out of the other buffer. This allows application code to execute in parallel with I/O transfers.

UNIT RECORD DEVICE FILE CONVENTIONS

Unit record devices (card readers, card punches, printers, terminals, and paper tape reader/punches) are used only for reading and writing data. They are not used for storing data, and thus do not require conventions for file identification and location.

Unit Record Device Pathname Construction

The pathname of a unit record device consists of the symbolic device name defined at system building preceded by an exclamation point (!). The pathname format is:

```
!dev_name
```

where dev_name is the symbolic device name of the unit record device.

Unit Record Device Buffering

All printers and most interactive terminals are provided with one File System buffer. (The operator terminal cannot be buffered.) By providing a File System buffer, application code can execute in parallel with I/O transfers.

All printers and all terminals (except the operator terminal) have a tabbing capability through software that converts the tab into spaces. Default tabulation stops are set at position 11 and at every tenth position thereafter for the line length of the device.

UNIT RECORD READ OPERATIONS

When an application task issues a logical read to a File System buffered device, one of the following actions occurs:

- If the buffer is full from a prior anticipatory read, the data in the buffer is transferred into the application task's area and a physical I/O transfer (an anticipatory read) is performed in parallel with continued 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; for other read actions it is the same.

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.

Interactive Terminal

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 or by checking for the appropriate status return after a Read, the application will not be stalled if the terminal operator is not present at the time of the Read request. Instead, the application can continue to poll other terminals.

Immediately after the Open is complete, a physical connection is made while the application continues execution. Depending upon the language the application is written in (for example, FORTRAN or Assembly language), it may be able to check the status of the Open to see if a Read can be issued without stalling application execution. The File System issues an asynchronous anticipatory physical read when the status check following 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" (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 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 the application until the anticipatory read is complete.

To avoid stalling on a Read or to avoid status check looping to test the input buffer status, applications should put themselves in the wait state, thus making the central processor available for lower priority tasks.

After the Open, an application written in COBOL must issue Read requests. The application will be put in the wait state if it is executing I/O statements in synchronous mode. Otherwise, the COBOL run-time package performs status checks and returns a 9I status until successful completion. The program can either loop on the Read or continue other processing.

BUFFERED WRITE OPERATIONS

A buffered write operation to a unit record device works on behalf of the application program in the same logical manner as a read operation. 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 devices.

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 is moved into it again. The application can avoid stalling execution when writing to an interactive terminal by doing one of the following:

- Checking the status of the system buffer before issuing the Write to see if the interactive terminal is still in use.
- Testing for a particular status return after the Write.

If a Write call is issued while data is being entered into the system buffer (because of a Read), the following sequence of events takes place:

- The read is allowed to complete.
- Input data is saved in the system buffer.
- A synchronous write is reissued by the File System.
- Output data is transferred directly from the application buffer.

Note that tab characters are not expanded into spaces.

If a physical I/O error occurs while data is being transferred from the system buffer to the device, you must be aware that the error occurred on the previous write operation. Furthermore, if any type of error occurs, the application program may need to have saved (or be able to retrieve) the data record so that it can be repeated.

Section 3
SYSTEM ACCESS

You can request access to the system to perform a number of different functions, such as:

- System building - Configuring the system to the needs of its users.
- System administration - Registering users.
- Operation control - Starting up the system each day, controlling processing, managing peripheral devices, and monitoring system status.
- Program development - Compiling, testing, and debugging programs.
- Application execution - Interacting with a program to accomplish a particular task.

In a large installation, different individuals will perform different functions. In a small installation, one person may perform most or all of the functions.

Access to the system is restricted to authorized users by means of the registration and login processes. Access to system files is restricted to specified users through the access control process (described in Section 2). Access to the various system facilities is controlled through the menu or command environment.

Before any other access to the system can be made, the system must first be configured.

SYSTEM CONFIGURATION AND DEFINITION

Creation of a system is a four-step procedure, consisting of:

1. Installing the operating system and application software onto your system disk. Note that this step is not necessary if you receive a disk with the operating system and application software already installed.

2. Bootstrapping a Honeywell-supplied system startup routine that provides a limited operating environment for building the files used in the third step.

3. Specializing 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 application programs.

4. Making a backup copy of the disk containing your specialized system.

Honeywell delivers to you one of the following:

1. A disk containing an installed operating system and requested application software. You use this disk to bootstrap the startup routine.

2. A LOAD/SAVE Diskette and a disk or tape containing the operating system and other application software you ordered. The LOAD/SAVE software guides you in installing the operating system onto your system disk. The installation process is described in detail in the Software Installation Guide. Once the operating system software resides on your system disk, you bootstrap the startup routine.

The bootstrap operation consists of turning on the power supply to the hardware, mounting the disk containing the MOD 400 system software, and pressing several keys on the control panel or System Control Facility device. (The procedure is described in the System User's Guide.) The bootstrap operation generates the initial configuration and startup operations. The resulting limited, one-user, on-line environment permits you to specialize system startup so that subsequent bootstraps will produce a multi-user environment that is adapted to your site's software requirements. This initial configuration may also be used to develop or execute application programs; however, the standard procedure is to generate a site-specific configuration first.

To generate a site-specific configuration, you either invoke the Autoconfigurator (DPS 6/22 only) or you create a file (called the CLM_USER file) containing the Configuration Load Manager (CLM) directives that describe the operating environment that will exist at your installation. The CLM_USER file is created automatically by the DPS 6/22 Autoconfigurator. The configuration directives are described in the System Building and Administration manual.

To further define the environment, you can create a START_UP.EC file (>>START_UP.EC) containing the operator commands that perform installation-specific functions such creating buffer pools. A >>START_UP.EC file is created automatically by the DPS 6/22 Autoconfigurator. (START_UP.EC files are described later in this section.)

After the CLM_USER file and the system START_UP.EC file are created, you again bootstrap the system. This time, the directives in the CLM_USER file control the configuration, and the operator commands in the system START_UP.EC file further define the operating environment.

It is important that you make a backup copy of the specialized system disk.

USER REGISTRATION

User registration is a process that protects the system from unauthorized access. Each person who is to be allowed on the system must be registered by the system administrator. The administrator uses the Edit Profile command to specify user-specific information such as:

COMMAND ACCOUNTING

Command accounting is an optional facility that logs all user commands entered through the command language, menus, and the Process Command Line system service macrocall. Commands entered from system groups (those whose first group id character is $) are not logged.

The system records the command's elapsed time and resource usage as well as the group id and user id of the issuer. Refer to the System Building and Administration manual for information on requesting command accounting and obtaining command accounting reports.

COMMAND BEAMING

Command beaming allows you to execute commands in another computer node. (A task group capable of processing the commands you enter is automatically created in the remote node.) When you issue a Beam command, the system's remote file access facility (described in Section 2) reads your command-in and user-in files and sends the data to the node specified in the Beam command. The output generated at this node is written to your user-out and error-out files. All subsequent commands you issue will be executed at the specified node until you issue another Beam command to return to your node.

All memory space, processor time, and disk space required to execute the commands are distributed to the remote node.

Since command beaming allows you into another computer, you can enter commands to find out the status of users, applications, devices, and so forth on that node. You can queue requests, send messages to local users, and update the node's remote file catalog.

EC AND START UP.EC FILES

The Command Processor is able to read commands from a source other than an interactive user terminal. One example is an Execute Command (EC) file that you construct through an editor. An EC file is a text file that contains command lines (for input to the Command Processor) and/or Execute command directives. An EC file is read by the Command Processor when:

- The Command Processor is invoked by an Execute command.
- A task group is activated with the Command Processor as its lead task and the EC file is specified as the task group's user-in file.

When you enter a request to have a task run in absentee mode, you specify an EC file that is to be read by the Command Processor (refer to the System User's Guide for further details).

EC Files

An EC file might contain a series of commands that you execute on a frequent basis, such as commands to execute a set of application programs that run at the end of the month to summarize inventory, sales, and accounts receivable. EC files can range from simple to complex. An example of a simple EC file is:

```
ED -PT
FORTRANA AREA -LE
LINKER AREA -IN LNKDR
DPRINT AREA.M
AREA
```

This EC file is made up of commands that are most often used in developing a FORTRAN program called AREA. The ED command invokes the line editor, FORTRANA invokes the the FORTRAN compiler, LINKER invokes the Linker, DPRINT prints the link map, and AREA executes the program.

A more complex EC file uses active functions and substitutable parameters. The following file could be used to create, compile, and link any program. (The lines beginning with the & character are Execute command directives.)

```
& CREATE, COMPILE, AND LINK A &1 PROGRAM
&P BEGIN EDITOR SESSION
ED -PT
&P COMPILATION BEGINS
&1 &2
&IF [EQUAL [RETCODE] 0000] &THEN &ELSE &> ERROR1
&P LINKER SESSION BEGINS
&A
LINKER &2
LINK &2
QT
&D
&IF [EQUAL [RETCODE] 0000] &THEN &ELSE &> ERROR1
&P LINK COMPLETE
&G FINISH
&L ERROR1
&P ERROR ENCOUNTERED IN DEVELOPMENT SEQUENCE
&P EC TERMINATED
&Q
&L FINISH
&Q
```

Assuming that the file is named PROG_DEV.EC, you could execute it file for a COBOL program development session by entering:

```
EC PROG_DEV COBOLA PAYROLL
```

The pathname PROG_DEV is substituted for all occurrences of &0 (none in this example), COBOLA is substituted for all occurrences of &1, and PAYROLL is substituted for all occurrences of &2.

EC files are discussed in detail in the System User's Guide and the Application Developer's Guide.

START_UP.EC Files

A special application of EC files is their use at system initialization and at task group activation.

SYSTEM START_UP.EC FILE

After configuration (after the CLM_USER file of configuration directives is executed), the system searches for a user-written command file named START_UP.EC in the system root directory. If this START_UP.EC file is present, it is executed. A typical system START_UP.EC file might contain operator commands used to establish an application environment for the installation. An example of such a START_UP.EC file is:

```
CBP BUFF1 -NBF 10 -CISZ 1024
CBP BUFF2 -NBF 5 -CISZ 512
CBP BUFF3 -NBF 5 -CISZ 256
CBP BUFF4 -NBF 20 -DIR
START_MAIL
EC >>GROUP$L
&Q
```

This START_UP.EC file creates several buffer pools of various common sizes, activates the local mail/message facility, and activates Listener monitoring of terminals.

USER START_UP.EC FILE

When a task group whose lead task is the Command Processor is activated, the Command Processor searches for an EC file named home_directory>START_UP.EC. (Home_directory is the pathname specified in the -HD argument of the user registration or the -WD argument of the Spawn Group command.) If such a file is present, the Command Processor executes it before performing any other action. This file could contain commands to direct the execution of the tasks of the job and/or perform certain housekeeping tasks. An example of a user START_UP.EC file is:

```
AMM -OFF
ST 2 -EFN PROGA
ST 3 -EFN PROGB
ST 4 -EFN PROGC
```

This START_UP.EC file causes three tasks to be activated and specifies that the user does not want to receive local messages.

There are important differences between tasks (and task groups) that are generated by a create function and those originated by a spawn function. Created task groups and tasks are permanent; they remain available in memory until explicitly removed. Spawned task groups and tasks are transitory; they perform a function and disappear.

Created task groups and tasks are passive; they must be explicitly requested to execute in order to perform their intended function. Spawned task groups and tasks cannot be requested. The spawning of a task group or task is equivalent to a create-request-delete sequence of control language commands. In a spawn operation, the task group or task is defined, provided with system resources and control structures, executes, terminates, and has its resources deallocated, all in one continuous process.

FORTRAN or Assembly task code may cause extensive action in its own behalf, as when application task code requests a system service or the execution of another task while awaiting the completion of the requested task. Each task that requests another supplies the address of a control structure through which the issuing task and the requested task can communicate, and which the Executive uses to coordinate task processing.

Task Group Identification

Each task group has a unique identifier. Honeywell-supplied system task group identifiers begin with a $ as shown in Table 4-2 below:

Table 4-2. System Task Group Identifiers

Task Group ID	Function
$H	Honeywell-supplied user task group
$L	Listener
$P	Deferred print
$S	System task group

The identifier for a user task group in the Create Group or Spawn Group command is a 2-character name that should not have the dollar sign ($) as its first character. The identifier (or group-id) can be indicated or implied in commands to designate what task group is to be acted upon. The operator can include the task group identifier when responding to operator terminal messages from the task group.

MEMORY MANAGEMENT AND PROTECTION

The system (hardware and software) provides a memory management and protection facility that performs the following functions:

- Allocates memory to guarantee each task group (user) its own address space.
- Protects multiple users from each other and the system from the users.

The hardware used to provide memory management and protection is called a memory management unit. The type of memory management unit varies according to the kind of processor. DPS 6 systems use either the Basic Memory Management Unit (BMMU) or the Extended Memory Management Unit (EMMU). Each of these memory management units is based on the concept of segmentation.

Segmentation

The memory management unit maps a segmented address space onto physical memory. The unit of memory allocation is a segment. A segment is a variably sized area of memory that usually consists of a logical entity such as a procedure. The system memory management and protection facility treats all addresses generated by the central processor as segment-relative addresses. It maps the segment-relative addresses through the memory management unit to absolute physical addresses. No segment can be less than 512 bytes in length. Segment size is always a multiple of 512 bytes.

SEGMENTATION WITH BASIC MEMORY MANAGEMENT UNIT

The BMMU supports up to 31 segments, 16 of which can be up to 8K bytes (K=1024) in size and 15 of which can be up to 128K bytes in size. The segments that can be up to 8K bytes are called "small segments"; those that can be up 128K bytes are called "large segments." The 16 small segments are numbered from 0.0 through 0.F; the 15 large segments are numbered from 1 through F. All small segments, and often some large segments, are reserved for system use; the actual number reserved is established at system generation.

The BMMU provides a total of 2 million bytes of segmented address space for each task.

Section 5
TASK EXECUTION

A task can be characterized as the execution of a sequence of instructions that has a starting point and an ending point, and performs some identifiable function. A task can initiate another task for execution or terminate itself by calling the task management commands or macrocalls. Multiple tasks can operate independently of and asynchronously to each other.

Each application, system, or device driver task operates at an interrupt priority level, one of the 64 priority levels provided for each central processor by the hardware and firmware. This section describes the processing of priority levels, including context saving of interrupted tasks and the assignment of priority levels and logical resource numbers to tasks. This section also describes task communication and coordination as well as deferred processing.

CENTRAL PROCESSOR INTERRUPT PRIORITY LEVELS

All system tasks, device drivers, and application tasks are assigned interrupt priority levels that indicate the order of their execution. This order of execution may be changed due to timeslicing (see below) or because this is a multiprocessor system.

Control of the central processor is given to the highest active interrupt level. However, in multiprocessor systems, a task at the higher priority may execute at the same time as a task of the lower priority since each task is executing on a different central processor.

Each central processor provides 64 potential interrupt priority levels that are used by the hardware to order the processing of events. These levels are numbered from the highest priority (level 0) to the lowest priority (level 63). Levels 0 through 5, 62, and 63 are reserved. The intervening levels (6 through 61) are assigned to logical resources (that is, devices and tasks).

The determination of which priority level is to receive central processor time is based on a linear scan of the level activity indicators. The level activity indicators are maintained by the hardware in four contiguous dedicated memory locations in each central processor (see Figure 5-1). Each bit that is "on" denotes an active priority level; each bit that is "off" denotes an inactive level.

Bits can be set "on" by software or by hardware events (interrupts). Most interrupting hardware devices are associated with priority levels during system configuration (by directives in the CLM_USER file). The three highest priority levels have dedicated assignments of special hardware/firmware functions such as incipient power failure, watchdog timer runout, and trap save area overflow. Priority level 3 is reserved as an inhibit level, level 4 is reserved for internal system use, and level 5 is dedicated to the real-time clock. Succeeding levels are user-configurable as device levels. Following these are three levels reserved for system use. Except for levels 62 and 63, the remaining levels can be used for application tasks. Level 62 is reserved for system use. Level 63 is reserved for an always active software idle loop or, in multiprocessor systems, for the task dispatcher.

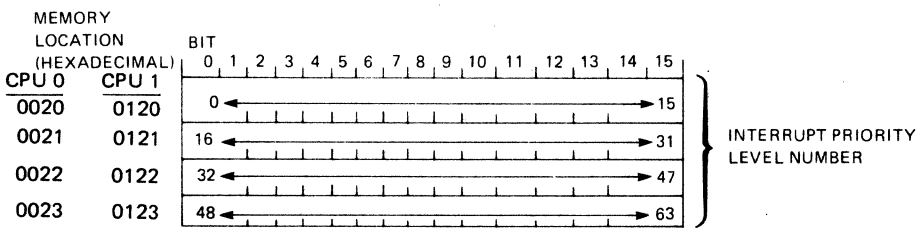


Figure 5-1. Format of Level Activity Indicators for each Central Processor

Multiprocessor Task Dispatching

In a multiprocessor system, the Executive maintains a queue of ready tasks ordered by priority level. This queue is called the general ready queue. The Executive dispatches the task at the top of the queue whenever a central processor becomes free to provide service. A dispatcher task runs at level 63 in each central processor and dispatches a task whenever it receives central processor time.

The dispatcher tasks attempt to balance the load so that high priority tasks are serviced before low priority tasks and all processors are used as fully as possible.

TIMESLICING

The technique of timeslicing minimizes the ability of user tasks that use large amounts of central processor time to interfere with interactive users of the system. In DPS 6 systems, timeslicing uses the Real-Time Clock Interrupt Servicing task (which executes at level 5) to check all tasks at a configured user level and below. Configuration of timeslicing is automatic. All user levels execute in a timesliced manner. Timeslicing options are discussed in the System Building and Administration manual.

The way timeslicing operates differs according to whether the system is monoprocessor or multiprocessor.

Monoprocessor Timeslicing

At each clock interrupt, a check is made to see if the task at the highest active user level has exceeded the configured value for a timeslice. (The system builder may specify the length of a timeslice in milliseconds or may accept the system default.) If the execution of that task has exceeded the timeslice value without waiting for some event, the task is removed from the front of the queue for its priority level and is placed at the end of that queue.

If a configured number of timeslices occur without the task waiting on any event, the task is demoted one priority level (that is, the task's priority level is increased by one). The task can be demoted again and again until it has been demoted the configured number of levels or has reached priority level 61.

Each time a task that was demoted waits for some event, it is promoted one level (that is, its priority level is decreased by one). The task can be promoted again and again until it reaches its assigned priority level.

Multiprocessor Timeslicing

At each clock interrupt, a check is made to see if the task at the highest active user level has exceeded the configured value for a timeslice. (The system builder may specify the length of a timeslice in milliseconds or may accept the system default.) If the execution of that task has exceeded the timeslice value without waiting for some event, the task is placed on the general ready queue as the last entry among tasks of its priority.

If a configured number of timeslices occur without the task waiting on any event, the task is placed on the ready queue and demoted one priority level (that is, the task's priority level is increased by one). The task can be demoted again and again until it has been demoted the configured number of levels or has reached priority level 61.

Each time a task that was demoted waits for some event, it is placed on the ready queue and promoted one level (that is, its priority level is decreased by one). The task can be promoted again and again until it reaches its originally assigned priority level.

TRAP HANDLING

The hardware provides a means by which certain events that occur during the execution of a task can be "trapped", with control being passed to software routines designed specifically to cover the condition causing the trap. Events such as the execution of a MOD 400 monitor call, or the detection of a program error, hardware error, arithmetic overflow, or uninstalled optional instruction cause traps (control transfers to designated software routines) to occur.

Traps are divided into two classes: (1) standard system traps, for which routines are supplied with the system, and (2) user-specific traps, for which users supply their own handlers.

An application program can designate which traps are to be handled by using the enable/disable user trap macrocalls (refer to the System Programmer's Guide - Volume II for details). If an enabled trap occurs in the user program, the Trap Manager transfers control to the connected trap handler for the condition causing the trap. A trap that is enabled is local to a task. Such a trap neither affects nor is affected by the handling of the same trap in another task, even within the same task group.

Any trap that occurs when its handler is not enabled, or that does not have a handler to process it, causes the executing task to be aborted.

SYSTEM FEATURES AFFECTING TASK EXECUTION

While MOD 400 does monitor resource use within a task group and among task groups, tasks and task groups must cooperate in their use of system resources to ensure smooth operation of the application.

Priority Level Assignments

Priority levels 6 through 61 are available for assignment to system, device driver, and application tasks. The system builder establishes the priorities of system tasks and driver tasks during configuration. (On the DPS 6/22, the Autoconfigurator establishes these priorities.) You assign the priorities of application tasks when you create task groups. Priority levels with low numeric values have higher priority than those with high numeric values. The procedures for establishing priorities are described below.

ASSIGNING PRIORITY LEVELS TO DEVICES AND SYSTEM TASKS

The system builder specifies hardware interrupt priority levels through an argument of the DEVICE directive in the CLM_USER file. (The Autoconfigurator is used on the DPS 6/22.) When the system builder specifies a particular type of device, the appropriate Honeywell-written device driver is loaded as part of the system. The three priority levels following the last one assigned to a configured device are used by system tasks and cannot be assigned to application tasks.

One example of priority level assignment is shown in Table 5-1. Levels 0 through 5 are assigned by the system and are not available to any user. The operator terminal is assigned to level 8; however, the system builder can assign any appropriate level to the operator terminal through a DEVICE directive. (The operator terminal must be at a lower (numerically higher) level than the Communications Supervisor.) At initialization, the system bootstrap device is assigned to level 6. This assignment remains in effect unless changed by a DEVICE directive.

Peripheral devices may be assigned to levels on both central processors in a multiprocessor system. This assignment is done automatically by the system.

Table 5-1 indicates Input/Output (I/O) devices, and not device drivers, to stress that each peripheral device must have at least one level assigned to it. Except for communications devices, peripheral devices cannot share a level. If there are two printers, each must be assigned a unique level even though there is only one copy of the associated I/O driver. Communications configurations require at least one nonsharable level dedicated to processing communications interrupts. This level must be higher than any level assigned to a communications device.

Communications devices can share a level. For example, four teleprinters (TTYs) and one Visual Information Projection (VIP) terminal can be configured to share one level or to use up to five levels. The priorities in Table 5-1 provide maximum throughput because devices with high transfer rates are assigned higher priorities than devices with low transfer rates.

Theoretically, the system builder could assign a level number as high as 58 to a device. In this case, levels 59 and 60 would be used by the system and only level 61 would be available for user task groups. In practice, however, the system builder would want to reserve more than one level for user task groups, especially for a system with a large number of devices. If priority levels 6 and 7 are assigned as shown in Table 5-1, the theoretical range of levels assignable through CLM COMM directives is 8 through 58. For a device associated with a COMM directive, the range is 9 through 58.

Table 5-1. Sample Priority Level Assignments for Tasks and Devices

Physical Priority Level	Base Priority Level	Use	Comments
0	N/A	Power failure handler	Levels 0 through 5 are automatically assigned by the system.
1	N/A	Watchdog timer runout	
2	N/A	TSA overflow	
3	N/A	Inhibit interrupts	
4	N/A	Reserved	
5	N/A	Real-time clock	
6	N/A	System bootstrap device	Set to level 6 at system initialization but can be changed.
7	N/A	Communications Supervisor	Must be higher level than any communications device.

Table 5-1 (cont). Sample Priority Level Assignments for Tasks and Devices

Physical Priority Level	Base Priority Level	Use	Comments
8	N/A	Operator terminal	Can be assigned any available level.
9 9 9	N/A N/A N/A	TTY device TTY device TTY device	Communications devices can share priority levels.
10 10	N/A N/A	Removable cartridge disk Fixed cartridge disk	The priority level for a pair of fixed/removable disks must be the same.
11 12 13	N/A N/A N/A	Diskette Diskette Diskette	
14 15	N/A N/A	Line printer Card reader	
16 17 18	N/A N/A N/A	Reserved by system Reserved by system Reserved by system	The three levels following the last device-assigned level are used by the system.
19 20 . . .	0 1 . . . 10	Task group A Task group B . . . Task group n	
. . .			
62 63	N/A N/A	Reserved by system System idle loop or task dispatcher	Always active.

ASSIGNING PRIORITIES TO APPLICATION TASKS

You assign priorities to user task groups and tasks when you create or spawn them. The command to generate a task group contains an argument that specifies the base priority level for the task group. The base priority level is relative to the highest number priority level assigned to a configured device. When a task group is assigned a base priority level of zero, the lead task of the group executes at the physical interrupt priority level that is three level numbers above the highest level number assigned to a configured device. 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 level at which a task executes is the sum of the following:

1. The highest level number assigned to a configured device plus 4
2. The base priority level number of the task group
3. The relative priority level of the task within that group.

This sum must not exceed 61.

Interactive user tasks are usually given higher priorities (lower level numbers) than absentee user tasks. Tasks that are I/O bound should be run at a higher priority than tasks that are central processor (CP) 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 CP-bound tasks. Otherwise, if the CP-bound tasks have a higher priority, the I/O devices would be idle while I/O-bound tasks waited to receive central processor time. (Timeslicing minimizes the ability of CP-bound tasks to interfere with interactive and I/O bound tasks.)

Logical Resource Number

A logical resource number (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

The system uses DEVICE directives to assign LRN values. Device LRNs may have values from 2 through 252, and from 256 through 4002. Figure 5-3 is an example of LRN and priority level assignments for devices and system tasks.

CREATING TASK GROUP REQUEST QUEUES

The operator uses the Create Group Request Queue command to create queue structures in which requests issued to a given task group will be stored. The operator must also issue a Start Mail command if one had not been previously issued. These procedures are described in the System User's Guide.

QUEUING TASK GROUP REQUESTS

You queue task group requests by issuing an Enter Group Request command. You can postpone action being taken on a request by specifying the -DFR (defer for interval) or -TIME (defer until date/time) arguments.

Once the operator has issued a Create Group Request Queue command for a task group, all further requests for that group are queued whether or not the requests are being deferred.

If the operator does not issue a Create Group Request Queue command, you can still submit group requests but will not be able to defer the requests.

Deferring Print Requests

The system provides a deferred printing capability under which your requests for printing specified files are queued in memory or disk mailboxes. The actual transcription of the files is done at a later time under the control of an operator-created system task group called a daemon.

After you submit a deferred print request, you can resume normal activities, log off, or reboot the system without losing the request.

The three steps involved in deferred print processing are creating the mailboxes, activating the daemon, and queuing the print requests. The information in the following paragraphs is conceptual. Detailed procedures for deferred printing are given in the System User's Guide.

CREATING PRINT REQUEST MAILBOXES

The operator establishes the mailboxes that are to contain the queued print requests. The mailboxes can be in memory or on disk. The mailbox names must be in the form $PR.Qn (n is an integer from 1 through 9 that identifies the relative priority of the queue, with 1 being the highest priority and 9 the lowest).

CREATING THE PRINT DAEMON

The operator is responsible for defining and activating the * daemon to process the print requests.

To create a daemon task group, the operator issues a Start Mail command (if one was not already issued), a Create Group command naming the daemon to be created, and an Enter Group Request command identifying the mailboxes to be used for queuing the requests and the devices to be used for printing.

Multiple daemon task groups can be run concurrently using common or separate sets of mailboxes and printers.

QUEUING PRINT REQUESTS

Once the daemon task group is active, you can queue print or punch requests by issuing Deferred Print commands. You can employ the -TIME argument to defer the printing of a file until a specified date and time.

Queuing and Transcribing Reports

Any file in print or punch format (i.e., any report file) can be queued and subsequently transcribed to an available printer or card punch. Report queuing and transcription is a spooling capability that provides automatic and manual report transcription, time-of-day printing or punching, and an automatic setup function that includes a sample transcription file (template).

The report queuing and transcription facilities control report transcription outside the context of the program. Reporting procedures for identical software can be totally different in different situations without requiring reprogramming.

Report queuing and transcription have three major aspects: creating a report queue, queuing a transcription request, and transcribing a report.

CREATING REPORT QUEUES

A report queue is a directory that allows you to place a report in a queue and subsequently transcribe the report. Report queues are created, modified, and deleted through Report Queue Maintenance (RQM) commands. The characteristics of the report queues are determined when the queue is created; the contents are determined when a report is placed in the queue for later transcription.

FILE BACKUP AND REORGANIZATION

File backup and reorganization is implemented through the Save and Restore utilities. The Save utility transfers disk files and directories to 1/2-inch or 1/4-inch magnetic tape or another specified storage medium. The Restore utility reconstructs the saved files and directories and puts them back on disk. Any file that has been saved and restored is automatically reorganized for disk space efficiency.

Since file access time efficiency may be lessened after a file has been in use for some time, it is recommended that disk volumes be periodically saved and restored. The files on the restored volume will be be compacted, resulting in optimal space allocation and improvements in the time required to search directories and check access rights.

Saving Files and Directories

The Save utility enables you to save an entire disk volume, a directory and all its subdirectories and files, or a specified file. If you are saving a directory, you can specify the number of levels of subdirectories (with their associated files) to be saved. Any access control lists associated with the saved files and directories are also saved, unless you specify otherwise.

The saved data, whether a whole volume, a file, or directories and files, is stored in a save file. The save file can be a magnetic tape or disk file, or an output device such as a card punch. When the Save utility processes the files and directories to be saved, it adds information that is meaningful only to the Restore utility. The saved files and directories are not just copies of the originals.

The Save utility can be executed while the files being saved are in use. Used with a journal file (refer to "File Restoration" later in this section), this type of save operation provides a dynamic and concurrent backup facility for high volume systems that cannot afford periodic shutdown to perform static file saves.

Restoring Files and Directories

You can restore from a save file all or part of the data you saved on that file. You can restore an entire volume (if you saved an entire volume), a directory and its associated subdirectories, or a specified file. Whatever you restore, you can return to the place from which you saved it, or you can place it in another directory or another volume.

Data saved from one type of disk can be restored to another type, provided the new disk has the required capacity. For example, you can restore a diskette volume onto a cartridge disk volume, or a partially filled mass storage device volume to a cartridge module disk volume.

POWER RESUMPTION

Power resumption is an optional facility that allows the system execution environment to be automatically restarted after a power interruption. The central processor must have the memory save and autorestart unit. This unit can preserve the memory image through a power failure lasting up to 2 hours. (It cannot, however, preserve the state of the I/O controllers nor can it ensure that no operational changes have been made to the mounted volumes.)

If fewer than 2 hours have elapsed when power is returned to the central processor, the power resumption facility will perform the following functions:

- Reinitialize the system software.
- Reconnect peripheral devices.
- Reconnect communication devices serviced by the Asynchronous Terminal Driver (ATD) line protocol handler or the Teleprinter (TTY*) line protocol handler. (Refer to the System Building and Administration manual and the System Programmer's Guide - Vol. I for information about these line protocol handlers.)
- Restart certain application tasks that were active at the time of the failure. Application tasks that are capable of being restarted are those using the display formatting and control facility and those containing user-written code to handle power failure and power resumption.

Implementing the Power Resumption Facility

The power resumption facility must be included in the MOD 400 Executive at system building. The central processor must contain a memory save and autorestart unit that has been activated by the operator (refer to the System User's Guide for activation procedures).

When power resumption is specified in the system building dialog, all peripheral devices and all communication devices associated with the ATD and TTY* line protocol handlers are designated as reconnectable and will be automatically reconnected when power is restored. If any ADT or TTY*-associated device is not to be automatically reconnected, the Set Terminal Characteristics (STTY) directive associated with the device must not contain the -RECONNECT argument.

GLOSSARY

HT (Horizontal Tab)

Command Processor: Reserved character.

Δ (space or blank)

Command Processor and Utilities: Reserved character; separates arguments and commands. Operator Interface Manager: At the beginning of a line, interrupts output.

! (exclamation point)

File System: A prefix indicating a physical device (sympd) name (for example, !LPT00). Line Editor: Escape character (for example, !F).

" (quotation mark)

Command Processor: Reserved character delimiting strings that contain embedded blanks (for example, "D. COOK"). See ' (apostrophe).

(number sign)

Line Editor: Signifies condition in If Data, If Range, and If Line directives. Linker: Specifies the current address.

$ (dollar sign)

Line Editor: In an address expression, represents the last line of the buffer (for example, $P). In any other Line Editor expression, represents the end of a line (for example, /DIVISION.$/). Linker: Specifies the next location (for example, BASE $). File System: First character of a macrocall name or mailbox (for example, $GTFIL).

% (percent sign)

Linker: Address argument representing the location two bytes greater than the highest address previously used in a linked root or overlay (for example, LDEF XTAG,%). Copy, Compare, Compare ASCII, and Rename Commands: Represents the character in the corresponding component and letter position of the entry name (for example, START_U%.EC).

& (ampersand)

Line Editor: Used in the string expression of the Substitute directive to indicate that the current expression is to be repeated (for example, S/TO BE/& OR NOT &/). Multi-User Debugger (numeric): Address symbol, representing the next address beyond the address used in the previous debug directive. Command Processor: Reserved character. Indicates continuation of a command on more than one line. Execute Command: Indicates EC directives and comment lines (for example, &P BEGIN LINK). TCL Compiler: Indicates continuation of a statement on more than one line.

' (apostrophe)

Command Processor: Reserved character. See " (quotation mark).

() (parentheses)

* Multi-User Debugger (Numeric): Delimits action lines to be stored for later use. Line Editor: Delimits multicharacter buffer name; optionally, delimits single-character buffer name (for example, B(EXEC)). TCL Compiler: Indicates insertion of field value.

* (asterisk)

Line Editor: In a regular expression any number of the preceding character. In a search directive, conditionality. CLM, TCL Compiler, Patch: Comment directive (for example, *SYSTEM DATA). File System: Represents one component of a file name (for example, COBPRG.*). In relation to Access Control Lists (ACLs) and Common Access Control Lists (CACLs), represents any person, account, and/or mode (for example, COOK.*.INT). List Profile Utility, Multi-User Debugger (Numeric): Signifies "all."

+ (plus sign)

Line Editor: Indicates addition to an address (for example, .+2P, /NEW/+3). Multi-User Debugger (Numeric): Performs addition.

, (comma)

Line Editor: Separates two addresses to be referenced inclusively (for example, 1,5P). CLM, Linker, Sort, and Merge: Separates arguments within directives.

- (minus sign)

Command Processor: Immediately precedes a control argument (for example, -ECL). Line Editor: Indicates subtraction from an address (for example, .-2P). Multi-User Debugger (Numeric): Performs subtraction.

. (period, decimal point)

File System: (1) Separates an entry name into components (for example, COBPRG.C). (2) Used as a single element at the beginning of a pathname to indicate the working directory (for example, .>FILE_DUMP). Line Editor: (1) In an address, represents the current line of the buffer (for example, .P). (2) In a regular expression, designates any character (for example, /PROG.AM/). Multi-User Debugger: Address symbol, representing the same starting address used in the previous debug directive. TCL Compiler: Indicates the end of a statement.

/ (slash)

File System: If first character of a star name, negates the meaning of the star name (for example, /*.WORK). See * (asterisk). Line Editor: Delimits strings in Expressions and Substitute directive (for example, S/OLD/NEW/). Patch and File Change: Immediately precedes a relative location or offset. Multi-User Debugger: Separates location from repetition value. Linker: Precedes a comment in a Linker directive file (for example, /SECOND OVERLAY).

: (colon)

Line Editor: Indicates label definition (for example, :7).

; (semicolon)

Line Editor: Separates two addresses; the first address becomes the current line, after which the value of the second address is calculated (for example, 2;.3P). Patch: Separates arguments in Patch directives. Sort and Merge: Separates directives. Linker: Separates Linker directives on one line. Command Processor: Reserved character. Separates commands. Multi-User Debugger: Separates directives.

< (less-than)

File System: Indicates movement in the storage hierarchy toward the root and a change in one level in that direction (for example, <LIBRARY). Assembler and Patch: Immediately precedes a relocatable address. Multi-User Debugger: Specifies the condition to be satisfied in an IF directive for conditional processing of the directive line.

= (equal)

Line Editor: Print Line Number directive. Multi-User Debugger (Numeric): Expresses equality for an IF directive. Linker: Address argument, specifying the base address associated with the object unit identified by an associated label (for example, BASE =OPNCRD). Copy, Compare, Compare ASCII, and Rename commands: Represents the corresponding component of a file name (for example, COPY FILE.A =.B).

> (greater-than)

File System: (1) Used at the beginning of a pathname to indicate a file or directory under the User Root Directory (URD) (for example, >SYSLIB2) and (2) Within a pathname, indicates movement in the storage hierarchy away from the root; connects two directory names or a directory name and a file name (for example, ^MYVOL>MYDIR>MYFILE). Line Editor: Go To directive (for example, >1). Multi-User Debugger: Specifies the condition to be satisfied in an IF directive for conditional processing of the directive line. Assembler and Patch: Indicates short displacement address. Execute Command: In a &> directive, go to first occurrence of the label following the current line.

>> (two consecutive greater-than signs)

File System: Used at the beginning of a pathname to indicate a file or directory under the System Root Directory (SRD) (for example, >>SID).

~ (tilda)

File System: Indicates the name of a magnetic tape volume. Allows a labeled tape volume to be referenced without specifying the name of a tape device.

? (question mark)

Line Editor: Address Prefix directive. Copy, Compare, Compare ASCII, and Rename Commands: Represents any character appearing in the corresponding component and letter position of a file name (for example, START_?P.EC). (See %.) File System and Command Processor: Immediately precedes a symbolic start address (entry point) in a bound unit name (for example, NOW?TIME). In some commands, requests help (for example, EP (Edit Profile)).

@ (at-sign)

TTY Terminal Driver: Delete the previously typed character (for example, TIMM@E).

[] (brackets)

Command Processor and TCL Compiler: Delimits active functions (for example, (CWD [USER NAME]). Multi-User Debugger (Numeric): Signifies the contents of the location defined by the expression within the brackets.

^ (circumflex)

File System: (1) Indicates a root directory, and must immediately precede a root directory name (for example, ^SYSRES) and (2) Used as a single element at the beginning of a pathname to indicate the root of the working directory (for example, ^>MYDIR). Line Editor: (1) When designated as the first character of a string, requests lines beginning with the string, excluding the circumflex (for example, /^IDENTIFICATION/) and (2) Indicates negation in certain directives. Multi-User Debugger (Numeric): Indicates negation as part of an IF directive.

_ (underscore)

File System: Joins two or more words in a file or directory name that the system is to interpret as one word (for example, LIST_PROG).

(pound sign)

Line Editor: If Data directive.

abbrev, login

See login abbreviation

abbreviation file

A file containing user-defined abbreviations and the character strings they represent.

abbreviation, login

See login abbreviation.

abbreviation processor

A system component that expands abbreviated commands and passes them to the Command Processor.

abort

An operator action resulting in the immediate cessation of operation of a task group or the operation of the currently executing request in a task group. All resources are returned to the Executive. The bound unit of the lead task of an aborted request may be retained.

absentee

A processing mode characterized by the absence of interaction between you and the system during execution of your program.

Access Control List (ACL)

A list specifying which user(s) can use the resource with which the list is associated.

ACL

See Access Control List.

activate

An operator action resulting in the resumption of a previously suspended task group. (See Suspend.)

active

A task is in the active state when it is executing or ready to execute, when its priority level becomes the highest active one in the central processor.

active function

A form of a command whose output string is placed in the command line before the rest of the line is processed.

active level

The priority level currently in effect.

address, absolute

A reference to a storage location that has a fixed displacement from absolute memory location zero.

address, relocatable

A reference to a storage location that has a fixed displacement from the program origin, but whose displacement from absolute memory location zero depends upon the loading address of the program.

administrator, system

Person responsible for registering users so that they can access the MOD 400 system.

after image

The image of a record in a restorable disk file as it exists after alteration. Written to a system journal file.

algorithm

A set of well defined rules for the solution of a problem.

alternate index organization

Alternate indexes are used to view a file ordered with a different key. The same data file can be ordered in many different ways by having more than one alternate index.

application program

A user-written program for the solution of a business, industrial, or scientific problem.

area

A DM6 I-D-S/II integrated file.

argument

User-selected items of data that are passed to a procedure (for example, system service macrocall arguments that are passed to the called system service, or command arguments passed to the invoked task). Synonymous with arg. (See parameter.)

argument, control

A keyword whose value specifies a command option. (See keyword.)

argument, positional

An argument whose position in the command line indicates to which variable the item of data is applied.

ASCII (American Standard Code for Information Interchange)

The interchange code established as standard by the American Standards Association.

asynchronous

Without regular time relationships. As applied to program execution, unpredictable with respect to time or instruction sequence.

attribute, file

Any of a set of disk file characteristics established when the file is created or modified to include such integrity features as recovery, restoration, and record locking.

Autoconfigurator

The DPS 6/22 system configuration utility.

base level

(See priority level, base.)

BCB

(See Buffer Control Block.)

BCD

Binary-Coded Decimal notation.

before image

A copy of a record from a recoverable disk file, as it exists just prior to updating, written to a system recovery file.

Binary Synchronous Communications (BSC)

A communications procedure, using a standardized set of control characters and control character sequences, for the synchronous transmission of binary-coded data.

form

A display terminal screen that provides areas (fields) into which you enter information that defines a function to be carried out.

full duplex

Simultaneous independent transmission of data in both directions.

full pathname

An absolute pathname which, when specified, begins with a circumflex (^) (for example, the root directory.)

function

A procedure that returns a single value to its caller. (See subroutine.)

globally sharable bound unit

A bound unit containing reentrant code and linked with the GSHARE directive. A globally sharable bound unit is loaded in the system pool and can be used by any task in the system.

group control block

A system structure describing attributes of a task group.

group_id

(See task group identification.)

group system space

An area of memory (segment) that contains the system control structures used to support a task group and its tasks in a swap pool.

group work space

An area of memory (segment) from which tasks in a swap pool obtain blocks of memory.

GRTS

General Remote Terminal Supervisor.

half duplex

Transmission of data in one direction at a time.

High Memory Address (HMA)

The address of the highest physical memory location in the central processor.

HMA

See High Memory Address.

hold file

A file that contains a copy of the Level 2 or Level 4 error logging statistics that are stored in memory. The hold file can be retrieved after system shutdown or crash.

home directory

Your initial working directory after logging in.

hot restart

Restart during a session.

I pool

(See independent memory pool.)

IMA

(See Immediate Memory Addressing.)

Immediate Memory Addressing (IMA)

A form of addressing a location in main memory by referencing the location directly, indirectly, or through direct or indirect indexing.

independent memory pool

A fixed partition memory pool. All tasks executing in a specific independent memory pool share a common virtual view consisting of all memory assigned to that pool and system global memory.

indexed file organization

A disk file whose records are organized to be accessed sequentially in key sequence or directly by key value.

indirect extent

The field in a directory record that holds the relative volume number that contains the succeeding set of extents.

Physical Input/Output (PIO)

Physical Input/Output, or physical I/O, that is initiated through a request I/O macrocall, outside of the File System, using IORBs.

PIO

(See Physical Input/Output.)

pipe

A special kind of sequential file used for synchronizing and passing information among multiple cooperating tasks.

pool identifier

A two-character name, established a system configuration, by which a memory pool is identified, and by which a task group is assigned a memory pool when the task group is created.

positional argument

(See argument, positional.)

power resumption

A system facility that controls the restarting of the execution environment following a power failure.

primary login

The form of login that requests Listener to spawn a task group that has the terminal from which the login originated as its primary system file (the terminal will be the initial user-in, command-in, error-out, and user-out files).

priority level

A numeric value that can be assigned to a task or device for purposes of controlling processing. Values range from 0 to 63. The lowest values (highest priorities) are reserved for system tasks; level 63 is the system idle level. Intermediate levels are available for user assignment to tasks and devices. The physical level at which a task executes is the sum of the highest level number assigned to a configured device plus four, the base level of the task group, and the relative level of the task within the group.

priority level, base

The priority level, relative to the system priority level, at which all tasks in a task group execute. A base level of 0 is the next higher level above the last (highest) system priority level.

priority level, hardware

A numeric value from 0 through 63 that can be assigned to a task or device to control processing. The lowest values (highest priorities) are reserved for certain system tasks. Level 62 is reserved for user tasks. Level 63 is the system halt level.

priority level, physical

(See priority level.)

priority level, relative

The priority level, relative to the base level, at which a user task within a task group executes. Relative Level 0 is the base level.

priority level, system

The priority level assigned to system devices and tasks.

profile

(See report queue profile file or user profile.)

program name suffixes

A "point-letter" character string such as ".O" for object units or ".A" for Assembly language source units appended to a file name to identify it as a source, object, or list unit.

protected string

A character string containing reserved characters that is enclosed by protected string designators. (See reserved character and protected string designator.)

protected string designator

A pair of quotation marks or apostrophes that enclose a character string containing reserved characters. (See reserved character.)

PVE

Polled Visual Information Projection (VIP) Emulator.

quarantine unit

A unit of message text; the smallest amount of transmitted data that is available to the receiver.

MOD 400 OPERATING SYSTEM MANUALS

Base Publication Number	Manual Title
HE01	ONE PLUS Guide to Software Documentation
CZ02	GCOS 6 MOD 400 System Building and Administration
CZ03	GCOS 6 MOD 400 System Concepts
CZ04	GCOS 6 MOD 400 System User's Guide
CZ05	GCOS 6 MOD 400 System Programmer's Guide - Volume I
CZ06	GCOS 6 MOD 400 System Programmer's Guide - Volume II
CZ07	GCOS 6 MOD 400 Programmer's Pocket Guide
CZ09	GCOS 6 MOD 400 System Maintenance Facility Administrator's Guide
CZ10	GCOS 6 MOD 400 Menu System User's Guide
CZ11	GCOS 6 MOD 400 Software Installation Guide
CZ15	GCOS 6 MOD 400 Application Developer's Guide
CZ16	GCOS 6 MOD 400 System Messages
CZ17	GCOS 6 MOD 400 Commands
CZ18	GCOS 6 Sort/Merge
CZ19	GCOS 6 Data File Organizations and Formats
CZ20	GCOS 6 MOD 400 Transaction Control Language Facility
CZ21	GCOS 6 MOD 400 Display Formatting and Control
CZ22	GCOS 6 VISION Reference Manual
GZ13	GCOS 6 MOD 400 R3.1 to R4.0 Migrataion Guide
HC01	GCOS 6 MOD 400 Application Development Overview

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: DPS 6
GCOS 6 MOD 400
SYSTEM CONCEPTS
ADDENDUM A

ORDER NO. CZ03-01A

DATED September 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: DPS 6
GCOS 6 MOD 400
SYSTEM CONCEPTS
ADDENDUM A

ORDER NO. CZ03-01A

DATED September 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell