Transaction Processing Facility

Concepts and Structures

Version 4 Release 1

GH31-0139-12

Transaction Processing Facility

Concepts and Structures

Version 4 Release 1

GH31-0139-12

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices” on page xiii.

Thirteenth Edition (June 2002)

This is a major revision of, and obsoletes, GH31-0139-11 and all associated technical newsletters.

This edition applies to Version 4 Release 1 Modification Level 0 of IBM Transaction Processing Facility, program number 5748-T14, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

IBM welcomes your comments. Address your comments to:

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures ix

Tables xi

Notices xiii
Trademarks xiii

About This Book xv
Before You Begin xv
Who Should Read This Book xv
How This Book is Organized xvi
Conventions Used in the TPF Library xvi
Related Information xvii
IBM Transaction Processing Facility (TPF) 4.1 Books xvii
IBM Enterprise Systems Architecture/370 (ESA/370) Books xviii
IBM Enterprise Systems Architecture/390 (ESA/390) Books xviii
Miscellaneous IBM Books xviii
Online Information xviii
How to Send Your Comments xviii

Introduction to the TPF System 1
TPF System History 3
TPF System General Applicability 5
TPF System Overview 5
TPF Production System 5
Supporting Environment 7
System Backup and Recovery 7
TPF Online System Elements 8
Main Supervisor 8
Database Support 9
Communications Control 10
Transaction Defined 11
TPF Transaction Services 12
TPF Processing Assumption and Performance 13
Benchmark Messages 13
Response Time 14
System Throughput (Messages Per Second) 15
Summary of the Meaning of TPF Performance 17

TPF System Processing Milieu 21
TPF System Parallel Processing 21
Multiprocessing and Multiprogramming 21
Concepts of Parallel Processing 22
Deadlock 23
TPF, ESA/370, and ESA/390 Architecture 24
The ESA Configuration 24
Central Processing Complex (CPC) 37
TPF System Program Structures 39
Application and System Programs 39
Reentrant Programs 39
Serially Reusable Programs 42
Multiprogramming Defined 43
TPF System Tightly Coupled Multiprocessing 44

Processor Lock . . . 44
Application Locks . . . 47
System Program Structures . . . 47
Performance Implication . . . 49
TPF System Loosely Coupled Multiprocessing . . . 49
TPF System Coupling Facility Support . . . 50
Coupling Facility Record Lock Support . . . 50
Logical Record Cache Support . . . 51
Multiprocessing and Multiprogramming Observations (Summary) . . . 51

TPF System Structural Characteristics . . . 53
TPF System Control Diagrams . . . 53
The TPF System Programming Terminology . . . 56
Control Structure for the TPF System Defined . . . 56
Message Processing Overview . . . 57
Execution Summary . . . 58
System Initialization . . . 60
CPU Loop (Dispatching Work) . . . 61
Operation Zero Program (OPZERO) . . . 62
Communications Source Program (COMM SOURCE) . . . 62
Message Flow Through the TPF System . . . 63
Step 1. The System is Initialized . . . 66
Step 2. CPU Loop Checks for Work on the Cross, Ready, and Input Lists 67
Step 3. Input Messages Arrive . . . 67
Step 4. Create an ECB and Select an Application . . . 67
Step 5. Fetch Application Program from File . . . 68
Step 6. Starting Program . . . 68
Step 7. Running Applications . . . 69
Step 8. Sending the Reply . . . 69
Step 9. Release Resources and Cleanup . . . 69
Summary of Message Flow . . . 69
Entry Control Block (ECB) Overview . . . 69
Format of an ECB . . . 70
Accessing the ECB . . . 70
Creation of an ECB . . . 70
Data Event Control Block Overview . . . 71
Main Storage Management Overview . . . 72
Virtual Address Space . . . 72
Fixed Storage and Working Storage . . . 73
Types of Dynamically Allocated Storage Available to an Application . . . 75
Dispatching (CPU Loop List Processing) . . . 76
Dispatch Control List (CPU Loop List) Management . . . 76
Enter/Back (Program Linkage) . . . 78
Program Nesting . . . 80
TPF System Program Classifications . . . 80
Control Program . . . 80
ECB-Controlled Programs . . . 83
TPF System Control Transfer . . . 83
Action on the Cross List (Switching I-Stream Engines) . . . 84
Switching an Entry to Another I-Stream Engine . . . 86
Switching I/O Processing Between I-Stream Engines . . . 87
Create Entries with Create Macros . . . 88
Common I/O Handler (CIO) . . . 89
File Storage (DASD) Accessing . . . 89
TPF System Magnetic Tape Support . . . 90
Unit Record Support . . . 91

Console Operations 91
Error Recovery 92
Entry Termination (EXIT Processing) 92
TPF System Structural Characteristics Summary 93

Data Organization 95
Database Overview 95
Multiple Database Function (MDBF) Overview 98
Fixed Records 99
Pool Records 99
Use Fixed Records and Pool Records 100
Data Record Attributes 102
  Physical Residence 102
  Logical Device Type 102
  Record Size 102
  Record Duplication 102
  Record Longevity 103
  Pool Record Types 103
  Types of Fixed File Records 103
Record IDs 104
  Record ID Attribute Table (RIAT) 104
Record Addressing 105
  Record Addressing Conversion Services (FACE, FACS, FACZC, and FAC8C) 105
  File Address Compute Table (FCTB) 105
  Application Record Addressing 106
  Record Accessing 106
  File Address Reference Format (FARF) 106
Record Holding 108
Module File Status Table 110
Record Allocation 112
  Relationship Between DBON and Physical Address 115
  Record Mapping 118
Duplication of Records 121
Pool Directories 122
Pool Management 124
  Pool Section 124
  Pool Segment 125
  Pool Directory 126
  Get File Storage 128
  Release File Storage 128
  Ratio Dispensing 128
  Pool Fallback 129
  Directory Reordering 129
  Short-Term Pool Recycling 129
  Pseudo Modules 129
Multiple Database Function (MDBF) 129
  File Address Compute Table (FCTB) 130
  Record ID Attribute Table (RIAT) 131
  Module File Status Table (MFST) 131
  Routing Control Application Table (RCAT) 131
  Global Area and Global Records 132
  Summary of MDBF 132
Unique Records and Shared Records 132
  Shared Records — Subsystem User 132
  Shared Records — I-Stream Engine 132

Shared Records — Processor 132
Unique Records — Subsystem User 133
Unique Records — I-Stream Engine 133
Unique Records — Processor 134
Basic Subsystem (BSS) 135
Switch Among Subsystems and Subsystem Users 135
Retain Module Records in Main Storage 137
Virtual File Access (VFA) 138
Globals 139
Retain Module Records in Module Cache Memory 142
General Data Sets 143
General Files 143
Loosely Coupled Multiprocessing — A Database Perspective 144
Record Hold Table and XLF Lock Table 144
Database Utilities 144
File Capture and Restore 145
Database Reorganization 145
File Copy 147
File Recoup 147
Pool Directory Generation and Maintenance 148
Database Generation 148
File Layout 148
File Allocation 148
Fixed File Record Initialization 148
Disk Module Initialization 149
Disk Module Formatting 149
Data Loading 149
TPF Database Facility (TPFDF) 149
TPF File System Support 150
Differences between Stream Files and Database Files 150
Using Stream Files in Programs 151
Directories 151
Path Name 153
Link and Symbolic Link 154
TPF File System File Attributes 157
Special Files 157
TPF Collection Support 159
Benefits of TPFCS 159
TPFCS Database 160
Cursors 165
Database Integrity 165
Database Archives 166
TPFCS APIs 167
Maintaining TPFCS 167

Data Communications 169
Functions of Communications Control 170
Message Routing Overview 170
Evolution of Communications Control 171
A Communications Overview of Message Processing 173
TPF Advanced Program-to-Program Communications (TPF/APPC) 183
TPF MQSeries Support 184
Local Queue Manager 184
Communication Interfaces 186
Error Recovery 186
Function Management Message Router (FMMR) 187

Interprocessor Communications (IPC) 187
User Exits 187
Transmission Control Protocol/Internet Protocol (TCP/IP) Support 187
Internet Daemon 188
Syslog Daemon 189
File Transfer Protocol (FTP) Server 189
Trivial File Transfer Protocol (TFTP) Server 189
Hypertext Transfer Protocol (HTTP) Server 189
TPF Internet Mail Server Support. 190
Remote Procedure Call (RPC) Server 190
TPF Internet Server Support 191
Storing Web Page Content in the TPF System 192
Retrieving Web Pages from the TPF System 193
Starting a TPF Application from the Internet 194

Index 197

Figures

1. Relationship of Chapters in Concepts and Structures xvi
2. A Processing Center 2
3. Airline Reservation Application 4
4. File Allocation Strategies 10
5. Response Time (per Message). 15
6. Throughput (Messages per Second). 16
7. Logical Structure of an ESA Configuration with Two CPUs 25
8. Page 0 Prefixing 28
9. Concurrent Interrupts 34
10. Interconnected Loosely Coupled Complexes. 37
11. Loosely Coupled Complex 38
12. TPF Process Structure. 41
13. Relationship of Locks 45
14. Simple System Control Structure Diagram 54
15. A Control Transfer Path 56
16. TPF Control Structure Diagram 57
17. Normal TPF System Execution Overview 60
18. Message Processing Flow Diagram 64
19. Entry Control Block (ECB) 71
20. Format of a DECB 72
21. Virtual Storage Layout 73
22. Action on the Ready List 78
23. Reentrant Stacks. 82
24. Multiple I-Stream Engine Processing Abstraction 86
25. Database Example 96
26. Record Allocation. 98
27. Multiple Database Function (MDBF) 99
28. Record Type and Ordinal Numbers. 101
29. FARF3 Format 107
30. FARF4 Format 107
31. FARF5 Format 108
32. FARF6 Format 108
33. Symbolic Module Numbers 111
34. Record Allocation Across Different Types of Module Devices 113
35. Total Record Space 114
36. Allocation Example. 117
37. Module to Module Duplication 122
38. Example of a Pool Directory 124
39. Pool Section 125
40. Pool Segments 126
41. Pool Directory 127
42. Relationship of FCTBs to Subsystems and Subsystem Users 131
43. MDBF: One Subsystem, Two Subsystem Users 133
44. Two I-stream Engines, One Subsystem 134
45. Loosely Coupled Complex 135
46. Cross System Access Services 137
47. Global Storage Allocation for a TPF Basic Subsystem with a Single I-Stream 141
48. Globals 142
49. Database Reorganization 146
50. Stream File and Database File Comparisons 150
51. TPF File System Directory Tree Example 152
52. Path Name Components 153
53. Identifying a File by a Link 154

54. Multiple Hard Links to a File 155
55. Using a Symbolic Link 156
56. General Layout of a TPFCS Database 161
57. TPF Communication Configuration 170
58. Communications Control Overview 174
59. Tables Used to Locate an Application 179
60. Destination — Output Message Routing 183
61. Socket Application Overview 190
62. Overview of TPF Internet Server Support 192
63. Storing Web Page Content in the TPF System 193
64. Retrieving Web Pages from the TPF System 194
65. Starting a TPF Application from the Internet 195

Tables

1. TPF Transaction Services Begin and End Transactions 12
2. TPF Transaction Services Suspend and Resume Transactions 12
3. Deadlock 23
4. Timing Sequence 30
5. Determining the Algorithm Relating the Cylinder, Head, Record, and Module to the DBON 119
6. Physical File Address Values When DBON=24 121
7. Comparison of Hard Link and Symbolic Link 156
8. Special Files and Their Associated Device Drivers 158
9. TPFCS Information 159
10. Collection Characteristics 161
11. Collection Type Summary 163

Notices

References in this book to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service in this book is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department 830A
Mail Drop P131
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Any pointers in this book to non-IBM Web sites are provided for convenience only and do not in any way serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this book or accessed through an IBM Web site that is mentioned in this book.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM
AS/400
CICS
DATABASE 2
DB2
Enterprise Systems Architecture/390
Enterprise Systems Architecture/370
EOCF/2
ESCON
IBM
MQSeries

OS/2
RISC System/6000
System/360
System/370
VTAM.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

About This Book

This book is a comprehensive technical overview of the Transaction Processing Facility (TPF) system. The book can also be used in an initial technical evaluation of the TPF system as a solution to a high-performance, transaction-driven, communications-based business system.

The Transaction Processing Facility (TPF) system is an Enterprise Systems Architecture (ESA) operating system that provides a responsive solution to online processing required in many business enterprises. In a TPF system, transactions are characterized by short messages that cause the instantaneous retrieval and usual modification of information related to business activity. The system has wide acceptance within the airlines industry for making seat reservations, but is also used for non-airline applications that require the use of terminals and workstations to access and modify information necessary to conduct a business or run an enterprise.

The TPF system emphasizes maximum performance. Performance means a response to the end user, who is an agent or customer of the business, within a few seconds or less. The unique system software is designed to accept very large transaction volumes from large populations of terminals and workstations attached through communication networks.

In this book, abbreviations are often used instead of spelled-out terms. Every term is spelled out at first mention followed by the all-caps abbreviation enclosed in parentheses; for example, Systems Network Architecture (SNA). Abbreviations are defined again at various intervals throughout the book. In addition, the majority of abbreviations and their definitions are listed in the master glossary in the *TPF Library Guide*.

Before You Begin

The reader needs a minimum of prerequisite knowledge but is presumed to have some experience with operating systems. The following list of terms is given to suggest the type of background that is assumed (however, you are not expected to know everything):

multiprogramming
SVC
start subchannel (SSCH)
interrupts
batch job
source code
DSECT
object code
virtual memory
multiprocessing
channel
linkage editor
operating system
load module
CSECT
data declaration
macro
dynamic address translation.

Who Should Read This Book

This book is intended for:

- Executives and technical personnel involved in evaluating the TPF system as a solution to a high performance, transaction driven, communications-based business system

- Systems programmers and engineers who have responsibility for installing, tuning, and maintaining a TPF system
- Application programmers who need or desire a greater familiarity with the basic concepts and methodology of the TPF operating system
- Systems programmers knowledgeable in other mainframe operating systems who desire to understand what distinguishes the TPF system from other operating systems.

How This Book is Organized

The organization of this book is such that an overview of the system is presented first, followed by chapters that provide additional detail for the concepts and functions introduced in the overview. Moreover, the attempt is to highlight how the TPF system differs from other transaction processing oriented operating systems. It is suggested that all readers start with “Introduction to the TPF System” on page 1.

The relationship of the different chapters in this book is shown in Figure 1.

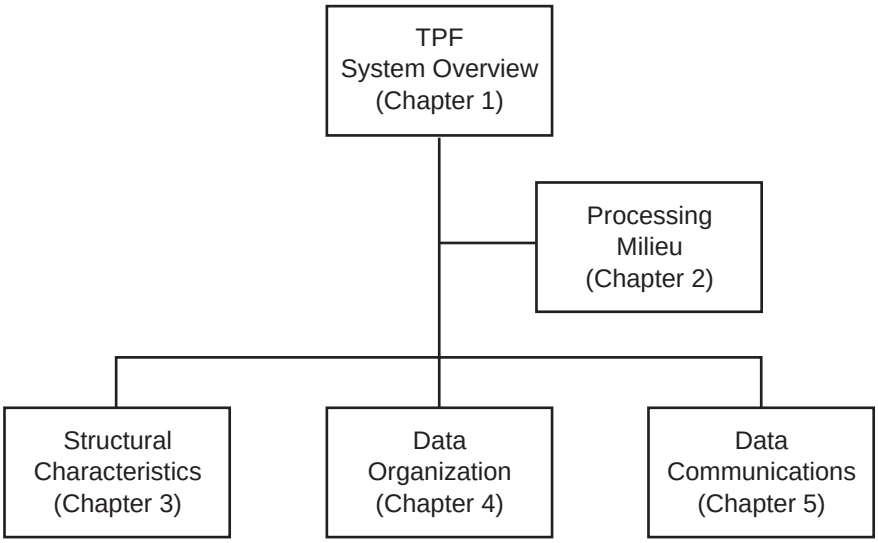


Figure 1. Relationship of Chapters in Concepts and Structures

Conventions Used in the TPF Library

The TPF library uses the following conventions:

Conventions	Examples of Usage
italic	Used for important words and phrases. For example: A *database* is a collection of data. Used to represent variable information. For example: Enter **ZFRST STATUS MODULE** *mod*, where *mod* is the module for which you want status.
bold	Used to represent text that you type. For example: Enter **ZNALS HELP** to obtain help information for the ZNALS command. Used to represent variable information in C language. For example: **level**

Conventions	Examples of Usage

monospaced	Used for messages and information that displays on a screen. For example:
	`PROCESSING COMPLETED`
	Used for C language functions. For example:
	`maskc`
	Used for examples. For example:
	`maskc(MASKC_ENABLE, MASKC_IO);`

bold italic	Used for emphasis. For example: You ***must*** type this command exactly as shown.
Bold underscore	Used to indicate the default in a list of options. For example: **Keyword=OPTION1** | **DEFAULT**
Vertical bar |	Used to separate options in a list. (Also referred to as the OR symbol.) For example: **Keyword=Option1** | **Option2** **Note:** Sometimes the vertical bar is used as a *pipe* (which allows you to pass the output of one process as input to another process). The library information will clearly explain whenever the vertical bar is used for this reason.
CAPital LETters	Used to indicate valid abbreviations for keywords. For example: KEYWord=*option*

Scale	Used to indicate the column location of input. The scale begins at column position 1. The plus sign (+) represents increments of 5 and the numerals represent increments of 10 on the scale. The first plus sign (+) represents column position 5; numeral 1 shows column position 10; numeral 2 shows column position 20 and so on. The following example shows the required text and column position for the image clear card.
	`|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...`
	`LOADER   IMAGE CLEAR`
	Notes:
	1. The word LOADER must begin in column 1.
	2. The word IMAGE must begin in column 10.
	3. The word CLEAR must begin in column 16.

Related Information

A list of related information follows. For information on how to order or access any of this information, call your IBM representative.

IBM Transaction Processing Facility (TPF) 4.1 Books

- *TPF Application Programming*, SH31-0132
- *TPF C/C++ Language Support User's Guide*, SH31-0121
- *TPF Database Reference*, SH31-0143
- *TPF Library Guide*, GH31-0146
- *TPF General Macros*, SH31-0152
- *TPF Migration Guide: Program Update Tapes*, GH31-0187
- *TPF Operations*, SH31-0162
- *TPF System Installation Support Reference*, SH31-0149
- *TPF Transmission Control Protocol/Internet Protocol*, SH31-0120.

IBM Enterprise Systems Architecture/370 (ESA/370) Books

- *ESA/370 Principles of Operation*, SA22-7200.

IBM Enterprise Systems Architecture/390 (ESA/390) Books

- *ESA/390 Principles of Operation*, SA22-7201.

Miscellaneous IBM Books

- *MQSeries Clients, GC33-1632*
- *MQSeries Distributed Queue Management Guide, SC33-1139*

Online Information

- *Messages (Online)*
- *Messages (System Error and Offline)*.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other TPF information, use one of the methods that follow. Make sure you include the title and number of the book, the version of your product and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send your comments electronically, do either of the following:
  - Go to http://www.ibm.com/tpf/pubs/tpfpubs.htm.

    There you will find a link to a feedback page where you can enter and submit comments.
  - Send your comments by e-mail to tpfid@us.ibm.com
- If you prefer to send your comments by mail, address your comments to:

  IBM Corporation
  TPF Systems Information Development
  Mail Station P923
  2455 South Road
  Poughkeepsie, NY 12601-5400
  USA
- If you prefer to send your comments by FAX, use this number:
  - United States and Canada: 1 + 845 + 432 + 9788
  - Other countries: (international code) + 845 + 432 +9788

Introduction to the TPF System

The TPF system provides an extremely responsive solution to very high volume online processing that is required in many business enterprises. The TPF system has wide acceptance within the airline industry but is also used in non-airline applications for processing relatively simple inquiry and response messages associated with a large population of terminals and workstations. The TPF system is most commonly used for the purpose of accessing a large centralized database that is an inventory of business information.

The TPF system is a high availability operating system designed to provide quick response times to high volumes of messages from large networks of terminals and workstations. A typical TPF system handles several hundred messages per second. A typical network varies from several hundred terminals and workstations to tens of thousands. The response time of the TPF system within a network is typically less than three seconds from the time the user sends a message to the time the user receives a response to that message. High availability is enhanced by the ability to quickly restart the system; restarting after a system failure takes between 30 seconds and two minutes.

These factors result in the unique design of the TPF system that can be summarized as:

- Efficient use of resources, such as main storage and file storage
- Short path lengths for critical system services, such as direct access storage device (DASD) input/output (I/O)
- Open-ended capacity growth, such as coupling as many as 32 multiprocessor Enterprise Systems Architecture (ESA) configurations with only a minimal increase in system overhead and expandable database capacity by the addition of direct access storage devices (DASD)
- High throughput while maintaining quick response times
- High availability allows for 24-hour, 7-day a week operation
- Database integrity and online database maintenance capability.

An ESA[1] configuration (see Figure 2 on page 2), used by the TPF system, incorporates multiple central processing units (CPUs) that are packaged together to share main storage. An ESA configuration is, therefore, defined as a channel subsystem and a set of CPUs that share main storage.

The TPF system also supports interconnected ESA configurations through the use of the ESA channel subsystem. In particular, multiple ESA configurations can be interconnected through ESA channel-to-channel (CTC) communication. In a fiber optic channel environment, an Enterprise Systems Connection (ESCON) channel, operating in CTC mode, supports the CTC communication.

The term *central processing complex* (CPC) is used within the TPF system documentation to denote an ESA configuration that is attached through locally attached channel subsystems to a set of devices or other ESA configurations. *ESA configuration* and *central processing complex (CPC)* are described in TPF System Processing Milieu. The computing facilities housed at a single location, which can

1. Within this publication, the term ESA means the hardware architecture described in the publications entitled *ESA/390 Principles of Operation* and *ESA/370 Principles of Operation* collectively referred to as Principles of Operation.

include interconnected ESA configurations, is viewed as a processing center, as shown in Figure 2.

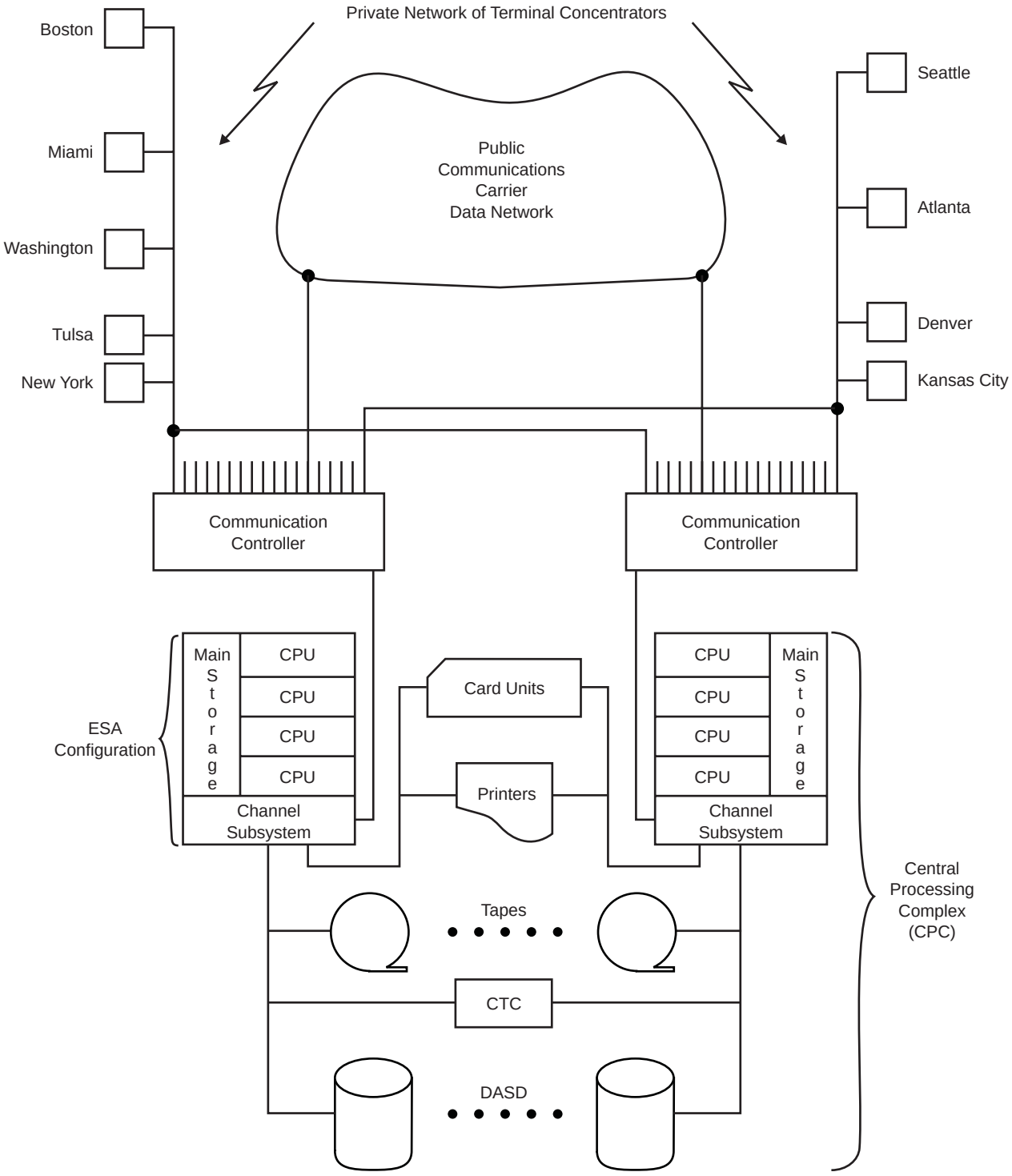


Figure 2. A Processing Center

Terminals and workstations are attached to a central processing complex (CPC) through wide area communication facilities. The phrase *wide area communication facility* means the use of transmission services provided by common communication carriers, which take two forms:

- Communications channels that are leased for the exclusive use of the TPF system, usually called *private lines*. Terminal concentrators are attached to these lines, forming a private network.
- Local access lines that are leased to attach to a common communication carrier data network that can be shared with other enterprises. Terminal concentrators are also attached to remote access points. (Remote is relative to a CPC; the terminal concentrators also attach through a local access line.) The management of the long distance routing and transmission, in this case, is the responsibility of the common carrier.

Processing centers can be linked together into networks through wide area communication facilities.

TPF System History

The origins of the TPF system can be traced back to systems created in the late 1950s to satisfy the requirements of airline reservation agents who accessed an inventory of available space on available flights for the purpose of selling tickets. A sale requires the deletion of a seat from the inventory of space and the creation of a passenger name record that accounts for the use of the seat.

This function, which is still a requirement today, is performed in real time and frequently occurs over two types of communication lines. A person representing an airline communicates with a customer by means of an ordinary telephone call, but accesses the necessary data using a terminal or workstation that is linked to a processing center through wide area communication facilities (see Figure 3 on page 4). The customer requesting a reservation is remote from the end user (airline agent) who in turn is remote from the data.

End users within the vernacular of the TPF system are commonly called *agents*, and the data usage requests they generate are called (input) *messages*. The important TPF characteristic persists: the ability to accept unpredictable and very high message volumes at a processing center that gives business agents access to a shared centralized repository of information (database) that is updated in real time.

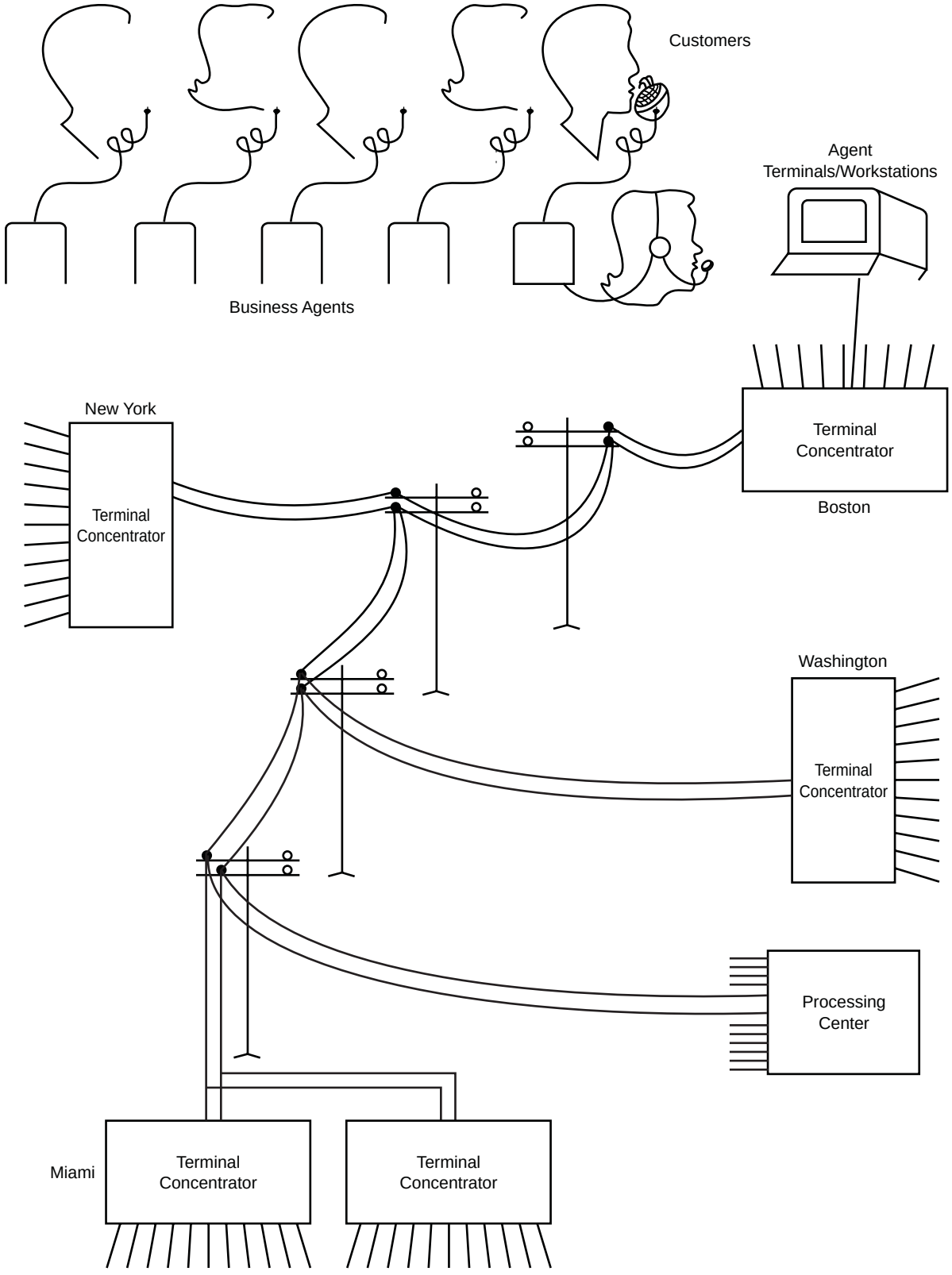


Figure 3. Airline Reservation Application

TPF System General Applicability

Any data processing environment that requires remote users to access shared information is a potential user of the TPF system. There are applications that can take advantage of the TPF system where the end user and customer are identical, for example, a person utilizing an automatic teller machine. Shared data, in some instances, may simply be the information necessary to route messages to appropriate locations within a network of processing centers. Some examples of non-airline applications utilizing the TPF system are:

- Hotel reservations
- Credit authorization/verification
- Police car dispatching
- Electronic funds transfer switching
- Online teller memo posting
- Message switching
- Loan payment processing
- Communication transaction routers.

TPF System Overview

This section introduces the various elements that must be considered when describing the TPF system.

- The TPF production system directly controls the management of the hardware resources assigned to the central processing complex (CPC) executing the TPF system, provides system services, and interfaces with communication networks. The hardware resources in a CPC are shown in Figure 2 on page 2.
- Non-TPF operating systems provide a supporting environment for the production environment for:
  - Program development of application programs
  - Assembly and compilation of programs necessary to install and maintain a TPF system
  - Running TPF system utilities; primarily related to the maintenance of the database in the production system.

  The nature of these functions does not require the high performance and high availability environment of the production system. Except for program testing, these functions are performed using MVS.
- Because of the high availability requirement of a TPF production system, the inclusion of a plan for system backup and recovery is insurance for the business of an enterprise.

Although the emphasis of this publication is on the TPF production system, an appreciation and understanding of the other elements in the TPF environment is a necessary prerequisite for installing and operating the production system.

Note: Historically, the TPF production system is called the online system and the supporting environment is called offline. These terms are pervasive throughout this publication and other TPF documentation.

TPF Production System

The production system is represented by system programs that manage the ESA hardware resources and interface with communication networks, as shown in

Figure 2 on page 2. The TPF main supervisor and related system programs are designed on the premise that work arriving at a central processing complex (CPC) arrives over data communication facilities in the form of a message. The term *Entry* is often used to describe an input message. Strictly speaking, an input message does not become an Entry until it is taken from an input queue and attached to an entry control block (ECB). Normally, an Entry processes one element of input data only. Thus, in contrast to batch processing systems that create tasks that can run for several hours, an Entry typically exists only for a few hundred milliseconds.

Note: Unfortunately, the term *entry* is often used generically in the TPF documentation to denote an item within a group of related fields in a table or list. Using the word *entry* in this context is easily confused with the TPF process construct called an *Entry*, described above. In this publication, this documentation dilemma is suppressed by using phrases such as *two-field item* or *work item*; further, the word *Entry* is capitalized when it is used to refer to the TPF application process.

There is no such thing as a job in the TPF system, and there is no need to create a teleprocessing (TP) subsystem or monitor to run within the framework of a batch operating system as, for example, the Customer Information Control System (CICS), which runs under MVS. The penalty for the TPF design: batch processing and computationally intensive work are not easily accommodated by the TPF main supervisor, and applications of this sort are seldom run under TPF control. The reward of the design: a boost in performance because interactive work does not incur the system overhead associated with a TP monitor in a general purpose operating system.

The TPF production system maximizes the number of messages that can be processed in a unit interval of time, usually expressed as messages per second, while maintaining rapid response times to the end users who enter the messages. Response time is measured as the time between user initiation of an input message and the display of the first character of the reply message. A TPF transaction implies several system responses in answer to a sequence of related user input messages. The TPF production system manages a large database in real time, which means the system continues to run even during periods of database maintenance.

The TPF system is characterized by:

- Real time inquiry/response from geographically dispersed users
- Relatively short messages (in terms of length) in both directions
- Unpredictable message arrival rate
- Shared business data
- Database update during real-time operation
- Database maintenance during real-time operation
- Database integrity
- Duplicate data records for performance and reliability
- A communications interface for supporting a very large population of various types of terminals and workstations
- Fast response time per message
- High availability (24-hour real-time operation)
- System restarts that require only 2 minutes or less
- Dynamic monitoring of system operation (system measurement).

Supporting Environment

The supporting environment is used to develop and test the applications that are run on the production system, as well as to generate and maintain the production system itself.

Applications that run in the TPF production environment are developed using high-level assembler, C, or C++ programming language. These applications are assembled or compiled in the supporting environment using the MVS operating system.

The requirements of test facilities depend on the extent to which an enterprise develops applications. For an enterprise that purchases TPF applications, the requirements for test facilities can be minimal. However, for an enterprise that develops applications, a test facility that closely resembles the configuration of the production system is often desirable.

A test system that provides a reasonable simulation of the production environment can be run:

- In a logical partition (LPAR), sharing a physical configuration with other operating systems running in other LPARs
- Under VM, sharing a physical configuration with other virtual machines
- On a dedicated configuration (this is sometimes referred to as stand-alone).

There are several aspects to the maintenance of the production system that must be considered:

- There is maintenance to the programs themselves to add function and correct code defects; programs are assembled and compiled in the supporting environment.
- The application database and some of the data for the production system are maintained through TPF system utilities, which are executed in the supporting environment.

System Backup and Recovery

System backup and recovery is the way in which an enterprise can recover its operation in the event of the unexpected. In today's business environment, the business of the enterprise is often heavily dependent on the availability of its TPF system, and so, it is necessary for an enterprise to understand, plan, and be prepared to prevent a single point of failure.

The planning and implementation of the hardware required for testing in a non-production environment is usually factored into an enterprise's business recovery plan.

A primary factor in planning and implementing business recovery is physical planning, and this ranges from what to do when there is a localized equipment failure (such as failure in a DASD controller or an ESA configuration) to a disaster (such as fire, earthquake, tornado, or act of terrorism).

In reality, enterprises that rely on a TPF system have plans that range from contracting with another enterprise to provide the physical resources to the other extreme where the enterprise has built into its physical environment precautions, such as locating the equipment underground and installing additional equipment. A compromise between these two extremes can be the sharing of equipment between a test configuration and the production system.

TPF Online System Elements

The user requests made of the TPF system are assumed to require very little computation, but considerable data manipulation. The online system is designed to prevent bottlenecks caused by queueing for system resources. The essence of the TPF system is to maximize performance for message driven applications on a given hardware configuration. Performance is viewed as a critical factor in the TPF system. This influences the techniques used to manage system resources. Generically, a control program or operating system defines application interfaces: control blocks, macros, supervisor calls, and so on. The TPF system has unique application interfaces that are influenced by techniques used to manage system resources.

The key elements of the TPF online system are:

- Processor control and support provided by the main supervisor
- Database support
- Communications control.

The functions provided by these elements are outlined in the following sections.

Main Supervisor

The TPF main supervisor provides the resource management usually expected of a control program in direct control of physical facilities. The precise techniques used to accomplish this resource management are topics in TPF System Structural Characteristics. The principal functions performed by the main supervisor are:

- Work scheduling

  Priority scheduling of input message processing is ***not*** done by the TPF system. An important processing concept that differentiates the TPF system from other operating systems is the way that it handles priorities and dispatches work (that is, an Entry).

  Work already in progress, that is, active Entries, has the highest priority. Scheduling is accomplished using a limited number of work queues. Priority is determined by processing all of the Entries on a queue ***before*** processing the Entries on the next queue. For example, the queue of work that represents completed I/O operations that were requested by active Entries (called the ready list) has higher priority than, and therefore is processed before, the queue of work that represents new messages arriving at a CPC (called the input list).

  In the TPF system, work scheduling is called *dispatching*.
- I/O interface

  In the TPF system, there are no access methods such as those found in MVS. Rather, channel programming is integrated into the system support of communication facilities, direct access storage devices (DASD), magnetic tape devices, and unit record devices.
- Virtual address space management
- Storage management
- Interrupt processing
- Keypointing

  Keypointing is the procedure by which system status is saved in the event that the system needs to be restarted because of a hardware or software malfunction. In other operating systems, this is usually called *checkpointing*.
- Error recovery.

Database Support

The structure of the online data is identified during system generation. Data is dynamically generated; however, the facilities for storing the data are identified prior to the online operation.

There are two organizations of file structures within the online data:

- The online database is the main database in the TPF system and its organization of data is unique to the TPF system. TPF files are allocated across the range of physical storage media to balance and, therefore, improve access performance. Therefore, successively numbered records are allocated to different physical direct access storage devices (DASD).

  A file request in the TPF system specifies a unique address whose exact form varies, but is always transformed into a record reference within an allocation scheme that spans the range of direct access storage devices (DASD), as shown by the horizontal allocation in Figure 4 on page 10.

  There are two categories of space within the online database:

  – A fixed file is used when the number of records required is relatively constant and the records are frequently accessed. A record in fixed file storage is used for a specific purpose.

    An application program references a record in fixed file storage through the use of a record type (which is a value) and ordinal number (a relative number of a record within a specified record type). The TPF system translates this reference into a symbolic address.

    A fixed file is analogous to a conventional multi volume data set organized for a direct access method. (Note that the term *data set* is an MVS term and is not used in conjunction with the TPF online database.)

  – The records in pool file storage, or just pool, are dispensed as needed and returned to the system when they are no longer needed and can be re-dispensed. When a pool record is requested by an application program, the system locates an available record and returns a pointer (address) to the pool record; typically the pointer is saved in a fixed file record for subsequent reference.

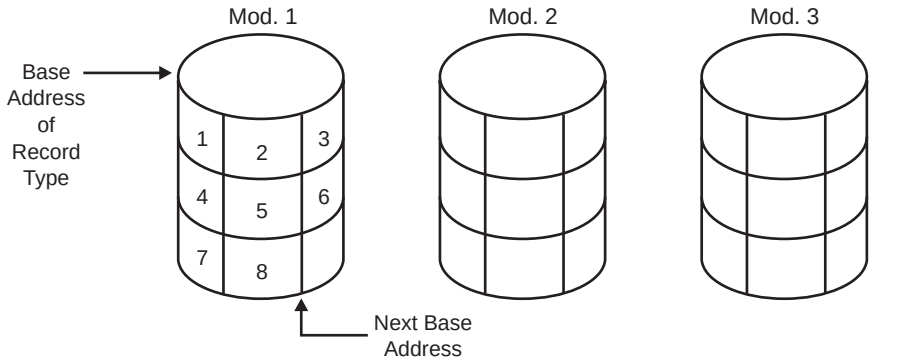
- Records in general file or general data set are contained on one physical storage medium and are allocated in a sequential fashion similar to a conventional MVS sequential data set. See vertical allocation in Figure 4 on page 10 for more information.

  This organization is used for several purposes:

  – The TPF system itself is loaded and started from a special general file, called the loader general file

  – Some of the online system utilities require a general file

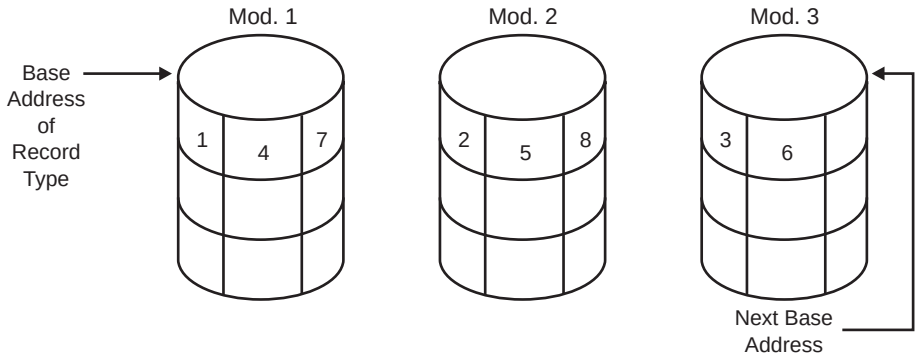
  – A general data set can be used for passing data between a TPF system and an MVS system.



Vertical Allocation:

Allocate a record type of 8 ordinal numbers to successive locations on a module

- The numbers represent the meaning of the next record within the record type. The meaning must be converted into physical characteristics, that is, module, cylinder, head, and record number.
- This allocation strategy is not used by the TPF system for the online database.



Horizontal Allocation:

Allocate a record type of 8 ordinal numbers to successive locations across all modules

- If each module represents an independent path to a CPU, then the accessibility to different records within the record type is improved.
- This represents the basic strategy used in the TPF system for the online database.

Figure 4. File Allocation Strategies

Communications Control

In the TPF system, communications control provides the interface to data communication networks and message formatting services for applications. Because of the historical evolution of the TPF system, the control and support of communications is divided into two distinct areas:

- Non-SNA communications control comprises the support from the very early versions of the TPF system. With this support, the TPF system manages (enabling, starting, polling, and so on) the communication lines in addition to formatting input and output messages.
- SNA communications control provides TPF interfaces with VTAM. An ACF/VTAM communications management configuration (CMC) residing in a central processing complex (CPC) other than the TPF system provides services such as resource and configuration ownership and management.

On input, a message is in a format that is specific to the device from which the message was entered. Because input messages can be entered from various types

of devices, the TPF system converts an input message with device-specific data to a common system format so that an application is able to process it regardless of its origin.

On output, the reverse is true; an output message must be converted from its common system format to be device-specific.

Additional detail about communications support is the subject of Data Communications.

Transaction Defined

Gathering the information needed for handling an item of business typically spans a few minutes. For example, two minutes per call is the average for the agents at one of the airlines using the TPF system. Typically, each piece of information is equivalent to a *message* in the TPF system. Each message represents an interaction between the TPF system and the agent. A group of related messages is, in the parlance of the TPF system, a *transaction*.

Several messages comprise a transaction, which results in processing of importance to the business enterprise, such as reserving an airline seat. The transaction required to reserve a seat on the flight can consist of information spread over several messages, as in the following example:

Message 1: Name of customer; phone number

Message 2: Date and time of flight; origin and destination

Message 3: Selection of desired flight

Message 4: Request for special meal service

Message 5: Indicate end of transaction

A transaction consists of one or more messages. Within the TPF system, a message is a component of a transaction. The number of messages required to complete a transaction depends upon the application design and the complexity of the specific information required by the end user. Credit verification applications, for example, often result in transactions of just one message.

Imagine, for example, that a two-minute (120-second) transaction consists of five messages and suppose the system responds to each message within three seconds. This means that only 15 seconds of the 120 seconds are spent ***within*** the system (where the system also includes the communications activity involved in transferring the message).

In general terms, this means that the time required to process a transaction is dependent upon the processing speed of the end user and the complexity of the transaction, while the response time for a single message is a function of the way in which the TPF system manages the computing resources.

Use of programmable workstations and message processing performed by front-end processors can affect the transaction mix by reducing the *think time* between messages and the number of messages transferred between the user and the host application. Programmable workstations can be used to perform preliminary editing of a message and to collect and package the messages in a transaction.

TPF Transaction Services

TPF transaction services support provides an interface for application programmers to ensure the integrity of the database. The TPF system uses a subset of the X/Open TX interface to begin and end a *commit scope*, which is a unit of work that groups together a set of database updates. With TPF transaction services processing, either ***all*** file updates have been completed or ***none*** of them have; that is, the updates can be either written to the DASD surface as a group at the same time or rejected as a group where no *hardening* (writing to the DASD surface) takes place. You never have to worry about a partially updated database.

An application programmer explicitly defines the start of the commit scope (commonly referred to as a *begin transaction*) and the end of the commit scope (commonly referred to as a *commit* or *rollback* transaction). Once a commit scope begins, updates to the database are stored in a commit scope buffer where they remain until the commit scope ends. A commit scope ends when the data is committed to the database or discarded (rolled back). None of the file changes are reflected in the database if the commit scope is rolled back. Following are the operations that permit the application to begin and end a transaction:

Table 1. TPF Transaction Services Begin and End Transactions

C Function	Assembler Macro
`tx_begin`	TXBGC
`tx_commit`	TXCMC
`tx_rollback`	TXRBC

Additionally, the TPF system provides the following extensions of the X/Open interface to the application to suspend or resume a transaction:

Table 2. TPF Transaction Services Suspend and Resume Transactions

C Function	Assembler Macro
`tx_suspend_tpf`	TXSPC
`tx_resume_tpf`	TXRSC

The activity in the commit scope does not affect activities outside the commit scope until the scope is committed or, in the case of nested commit scopes, until the root scope is committed. A *nested scope* occurs when an application programmer begins a transaction (opens a commit scope) before ending the previous commit scope. In this case, the file updates that are committed or ended by the nested scope are still not seen by any other ECBs until the root commit scope ends. The *root scope* is what the first, or highest level, commit scope is known as.

At the system level, TPF transaction services includes support for a transaction manager, resource managers, log manager, and recovery log to ensure a consistent view of the database.

- The *transaction manager* (TM) provides a set of application program interfaces (APIs) for the application to define both the scope of a transaction as well as actions to be taken for the transaction. The TM coordinates resource managers and determines which resources are written to the recovery log at commit time and which resources are recovered at restart time.
- The *resource managers* (RMs) work with the TM to identify and harden resources used by the application in a commit scope. TPF DASD and pool

support are the resource managers supplied by IBM. You can write your own RM provided that it is consistent with the architected TM and RM interfaces.

- The *log manager* controls the recovery log and recovery actions.
- The *recovery log* is written to DASD; it holds the data necessary to recover resources following a system failure without compromising the integrity of the database.

TPF Processing Assumption and Performance

Performance sensitivity is part of the fabric of the principal function offered by the TPF system, that is, processing messages that are requests for information from a large centralized database.

The TPF system is designed on the assumption that each of the end users generates messages (the component parts of transactions) that require only small or *trivial* amounts of CPU processing. CPU processing per message is trivial when compared with the delays inherent in the communication facilities and in accessing information from the large database. The term *trivial processing* is used in computer science literature to describe work units that require very little processing resource, which certainly characterizes the work units managed by the TPF system. However, since trivial processing is easily confused with an unimportant request, the phrase *expeditious processing* is used in this section to suggest prompt and efficient service of presumably important requests, each requiring very little CPU processing power. The meaning of performance within the environments using the TPF system dictates this expeditious processing assumption.

Performance within the TPF system means a designated response to a benchmark message at a designated message rate (given in messages per second). This aspect of performance is based upon the system response time to any end user message and not upon the length of time to complete a transaction. Of course, if the system response time fluctuates greatly, the performance of the end users is affected. Within a given installation, an example of a service level statement can be:

> "The system will be capable of providing a three second message response time to 95% of the end users during the intervals when the system is also processing 1500 messages per second. For any message rate, the response time is based upon an average message that causes 50 000 CPU instruction executions and 10 accesses to data on the physical DASD surfaces."

Benchmark Messages

The benchmark message is an important ingredient for making specific performance statements. A business enterprise contemplating installation of a TPF system must identify its benchmark message. This requires familiarity with the envisioned applications as well as familiarity with the TPF system structure. The number of file accesses, for example, depends upon an awareness of the TPF data base support; the number of instructions executed depends upon application processing as well as the instructions executed to provide system services such as obtaining messages from a communications network. The identification of a benchmark message is nontrivial and is a joint effort done by application and system personnel. The performance issues are only indirectly the subject matter of this publication. The subject matter of this publication is the system structure in support of the computing resources, which are influenced by performance issues.

Note: In this publication, when a message is discussed within the context of performance, it is a benchmark message.

Response Time

Response time is relative to an individual end user who is not interested in the resource utilization or how many other users happen to be using the system. It turns out that an end user is usually most interested in a response at the very same time that many other users are also interested in their responses. That is, all the end users are busy simultaneously as, for example, can be the case for bank tellers at various branches throughout a city during lunch hour.

A line representing response time in Figure 5 on page 15 shows the principal delays that contribute to the time needed to respond to a message. This figure is a representation of a message requiring a total elapsed time of 3 seconds from the moment an end user makes a request at a terminal or workstation until the reply begins to appear at the user's terminal or workstation; the total elapsed time is called *response time* in the TPF vernacular. The measurement of response time begins at the instant the end user enters a request (message) and includes the time that the message travels over communication lines to a communication controller, and arrives through the channel subsystem at a CPU in a central processing complex (CPC). At the CPC, programs are invoked to process the message by accessing a large database, formatting a reply message, and sending the reply. The time required for this to occur is also included in the response time.

Figure 5 on page 15 shows a CPU occupancy of 1/2 second or 500 milliseconds (ms). CPU occupancy includes the creation of a control block that identifies the programs and data necessary to perform the message processing. Although not all the programs and data need reside in main storage during the processing of the message, the control block does. During this CPU occupancy, most of the time is usually spent waiting for I/O to be completed and very little time executing CPU instructions. The CPU occupancy of any given message consists of several intermixed processing intervals and I/O delays. Because the I/O gaps represent most of the delay while the message is in the CPU, the number of channels to secondary storage, queueing disciplines, and the organization of data are very important in order to maintain fast response times at peak periods; these issues are related to minimizing the length of the I/O gaps.

The contribution of the processing intervals to the response time delay of a single message is very slight. A *very slight* processing interval, of course, depends upon the power of the CPU doing the processing. In the example response time line of Figure 5 on page 15, the I/O delays can account for 494 milliseconds and the processing intervals for 6 milliseconds. Clearly, if the system responds to only one user, the CPU can be orders of magnitude slower without any dramatic change in the single user's response. Multiprogramming is employed to utilize a single CPU on behalf of other messages for other transactions during the I/O delays for a given message. Multiprocessing is used to allow multiple CPUs to share the processing load of very high message rates.

TPF systems are generally used in environments where economies are realized by an affinity of large volumes of data with sufficiently powerful CPUs to service thousands of users at peak periods. The TPF system is used in some environments to hold shared network data in order to distribute processing function throughout a network of processing centers. For example, the system is used in credit verification applications to process a credit inquiry or to route the inquiry between an agent and an appropriate processing center.

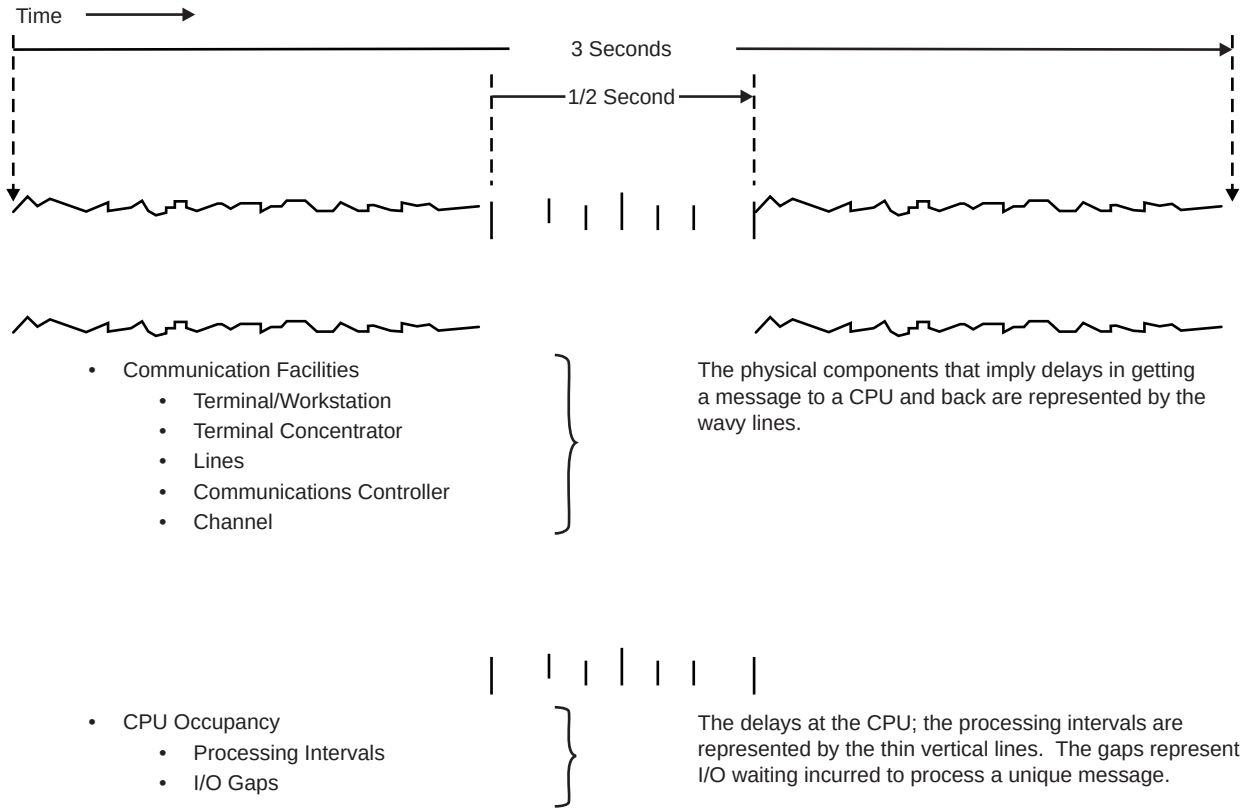


Figure 5. Response Time (per Message)

System Throughput (Messages Per Second)

The number of messages processed over a given interval of time is called *system throughput*. A business enterprise must identify its projected peak message rate in order to assess whether the TPF system is an appropriate solution or not.

The number of CPU instructions required to process a message is frequently called *path length*. The instructions include the application processing as well as system services, such as receiving the message, transmitting the response, accessing the online database, and processing interrupts. The path length of the response time line given in Figure 5 is represented by the thin vertical processing interval lines. The path length of a benchmark message is a composite, or average, of the path lengths of different kinds of messages actually processed in the system. The number of instructions executed each second by a CPU over a peak period is obtained by multiplying the path length of the benchmark message by the average number of messages processed each second during a peak period. Note that sufficient resources to deliver messages to the CPU must be available.

The instructions to be executed over the peak period is generally given in million instructions per second (MIPS), which identifies the processing power required of a CPU to handle the message rate. This, of course, can also be stated in terms of messages per second, a statement of throughput, not response time. The throughput requirement of a single CPU can be shown by the summation of a sufficient number of processing interval lines that all add up to the solid dark line in

Figure 6 on page 16. The solid dark line is obtained by summing the maximum number of messages that can be processed by a given CPU during the interval determined by the CPU occupancy of a single message. In the example of Figure 5 on page 15, this interval is given as 1/2 second. Therefore, the number of response lines in Figure 5 required to saturate the CPU (shown in Figure 6) represents the maximum number of messages that a given CPU can process in 1/2 second.

The relationship between MIPS and messages per second is simple:

```
MIPS = (Messages per Second * Instructions per Message) / 1,000,000
```

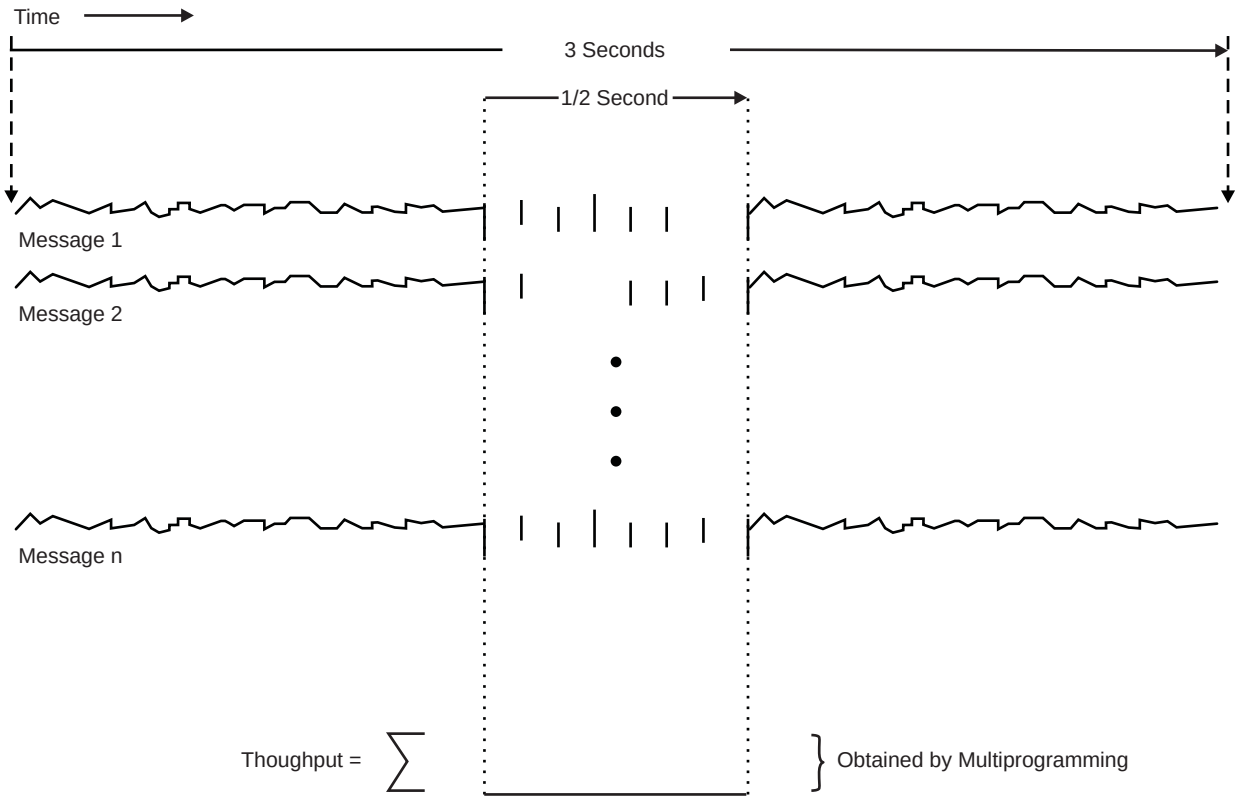


Figure 6. Throughput (Messages per Second)

An almost obvious fact from queuing theory shows that if all CPUs are utilized 100% of the time, there are messages waiting to be processed. This does not mean that a CPU at 100% utilization is not capable of delivering its peak capacity. It does mean that a message arriving at a processing center must *wait in line* for a CPU to become available. (As an example consider what happens to you during the busy lunch hour activity at a bank within a large city. In this case, you are the *message* and the teller is the *CPU*. Assume that fatigue has not overcome the teller, a fair assumption to be made about real CPUs.) The waiting time to get into the CPU is incorporated in the wavy line of Figure 5 on page 15. The waiting time becomes very large for every message at 100% CPU utilization. Therefore, a rule of thumb when planning a TPF configuration is to design for CPU utilization not to exceed a measured 85% during peak periods. This value allows for variations in the message load during peak periods and growth projection, as well as for deviations from your system plan.

This utilization is based upon a little theory and many years of observations. Thus, the MIPS in the previous formula are divided by .85, which has the effect of

increasing the necessary CPU power to deliver the required response time at the designated peak load. This should provide an idea of the relationship between response time and system throughput; both are related to performance but are in conflict with each other. (For example, you get better service at the bank if not all the tellers are busy; good for you; bad for the bank.)

An accurate path length of a message must include all CPU instructions executed. This translates to mean:

> *Every instruction executed must count against a message in an environment dedicated to the fast response time of each message during the periods of high volume message processing*.

The TPF system architecture is designed to keep the path length for message processing as short as possible. In other operating systems, the path length of a message increases at peak loads while the path length of a message in the TPF system is relatively constant. In other words, the instructions executed for system processing required at peak message rates are not much different than the instructions required at lower rates. The instructions executed for application processing are assumed to be constant. (In some non-TPF systems, the path length of a message increases significantly as the message load increases because of the complexity of handling additional control blocks.)

Summary of the Meaning of TPF Performance

Within the TPF system, performance means:

> *Fast* response to *expeditious* processing demands at *peak* message volumes. Fast, expeditious, and peak must all be identified by the customer with some assistance from a TPF consultant. The TPF system structure in support of a central processing complex (CPC) is designed to keep the processing on behalf of each message as trivial as possible. A TPF system configuration and the system software are influenced by the requirement for consistent response times at peak loads.

Messages per second represent the throughput component of performance. The TPF system incorporates architectural philosophy to handle the response time component of performance, as well as system availability. This may differ from operating systems, that do not emphasize transaction processing. Architecturally, response time is improved by:

- Minimizing the instruction path length for critical system services

  For example, the logic to access data is integrated in the system software that, in other systems, is frequently found in application appendages called access methods.
- Eliminating bottlenecks caused by queuing for access to shared system resources that are necessary for processing any message
- Minimizing the number of I/O requests
- Increasing the number of data paths available for processing I/O requests
- Allocating shared data resources prior to online operation
- Organizing the physical structure of the shared data to improve data accessibility.

  A user installation must customize this aspect of the system to match its unique application requirements.

The design of the TPF system is influenced by performance and availability requirements, and results in an architecture with the following characteristics:

- Consistently fast response times, high message throughput, and extremely high availability
- The length of any given system outage is minimized in contrast to minimizing the average outage over some operational period

  For example, multiple outages of one minute over a one month period may be acceptable whereas one outage of multiple minutes would be unacceptable.
- Data access techniques that are an integral part of the system software

  The system access techniques are structured to improve access paths to shared data and to minimize the amount of time required for system restarts.
- Storage management that allows for only pre-defined *block* sizes

  A *block* has roughly the same meaning as the storage management unit called a *page* in virtual storage systems.
- Collection mechanisms for performance measurements and system tuning that are an integral part of the system software

  The system collection of most of the variables necessary for performance measurement do not bias normal online operation because the collection is, in essence, analogous to a *gauge*, that is, operative whether or not measurements are analyzed. For some measurements, statistical samples are periodically taken throughout the interval of interest. Thus, the overhead that the measurement facilities introduce is almost indiscernible.
- A set of test tools to simulate the production environment is an integral part of the TPF system

  These tools are used to execute new and modified applications in a simulated environment. The test tools are used to ensure that applications adhere to interfaces in the TPF system. This eliminates redundant and costly system overhead for validity checking on each and every message in the production environment.
- An efficient interface to data communication facilities for accepting units of processing work that includes facilities for minimizing the length of a system restart.

  The units of work are called messages or Entries (not tasks or job steps). There is no such thing as Job Control Language (JCL) or a job entry subsystem (JES); the TPF analogue to JES is communications control in the TPF system.

The difference between the transaction environment in the TPF system and classical batch-oriented operating systems can usually be traced to the following statements about expeditious processing:

- The TPF system is designed on the assumption that there must be on-demand servicing of unscheduled but predictably trivial requests for CPU services in an environment of fixed resources.
- Many computing systems are designed on the premise that there is on-demand service of unscheduled and unpredictable requests for CPU services in an environment of fixed resources.

The TPF expeditious processing statement, statement #1, is based on the assumption that all work (individual messages) arriving at a CPU requires small or trivial amounts of the CPU resource. The non-trivial processing statement, statement #2, is based on the assumption that at least some of the individual requests require large amounts of CPU resources. As processing power becomes less expensive, many of the reasons for making the second assumption become less important, for example, provide a separate processor for each of the users involved in complex calculations. On the other hand, an increasing number of data

processing installations are used to allow business agents to access shared data resources, in which case the need for systems designed to the first assumption become more important. The required power of the processor, using the first assumption, is not a matter of computational complexity but the sheer volume of requests, each requiring only a small amount of the processor resources. A request that requires expeditious processing could be, “Get my powerful processor some shared data,” or “Get me to a powerful processor”.

Finally, although a TPF system has the capability to communicate with an external system, this discussion of performance applies only to messages that are processed by a CPU within a TPF system. There are other performance considerations when processing is performed by a system external to a central processing complex (CPC) even if the external system is TPF-based. A discussion of performance involving an external system is beyond the scope of this publication.

TPF System Processing Milieu

The origin of the TPF system can be traced to systems that were implemented in the late 1950s establishing application interfaces. Subsequent versions of the TPF system have been produced all of which preserve these application interfaces of the past. This also generally introduces constraints and compromises that would not exist if there was no installed base of applications.

The TPF system is easier to understand with some knowledge of the general computing principles employed, some knowledge of the ESA architecture, and some knowledge of the past. So, within the context of the past, this chapter presents a processing milieu that is necessary background for comprehending the TPF system.

TPF System Parallel Processing

The TPF system was originally designed on the assumption that programs execute on a single central processing unit (CPU), commonly called a uniprocessor. A synonym for CPU, used in the TPF publications, is *instruction-stream engine* or simply *I-stream engine*. An I-stream engine is just a CPU within an ESA configuration. Several instruction-stream engines can be combined into a single ESA configuration and can work either together or independently.

There are two senses of parallel processing enabled by this architecture. One sense considers two or more ESA configurations. In the TPF system, there is a facility for several ESA configurations to operate as a single complex, called *loosely coupled*. The other sense considers a single ESA configuration where multiple I-stream engines execute concurrently; this is called *tightly coupled*. Combining these two senses of parallel processing means that an ESA configuration running the TPF system in 1, 2, or as many as 16 I-streams, tightly coupled, can be tied together with other tightly coupled ESA configurations in a loosely coupled complex of up to 32 ESA configurations to yield the processing power of all these combined I-stream engines. Figure 2 on page 2 shows two ESA configurations, each with four I-stream engines (CPUs).

Tightly coupled multiprocessing refers to the synchronization of accesses to shared main storage in an ESA configuration of multiple I-stream engines. An ESA configuration with only one I-stream engine is called a uniprocessor, and one with multiple I-stream engines a multiprocessor. Uniprocessor and multiprocessor are terms within the TPF system that are associated with tightly coupled multiprocessing.

Loosely coupled multiprocessing involves two or more ESA configurations sharing a set of module (sometimes referred to as DASD) control units (CUs) along with an external lock facility (XLF) for synchronizing accesses to the module records by multiple ESA configurations. XLF is logic in the module CU or a coupling facility (CF) called by any ESA configuration attached to the module. This implies that all the participating ESA configurations are channel attached to the same module CU or CF.

Multiprocessing and Multiprogramming

System diversity provides multiprogramming and multiprocessing capabilities within the TPF system. Multiprogramming and multiprocessing are incorporated to increase the number of messages that can be processed over some interval of time, usually given in messages for each second.

- Multiprogramming means that several programs (sequences of ESA instructions), in different stages of execution, are coordinated to run on a single I-stream engine (CPU).
- Multiprocessing is the coordination of the simultaneous execution of several programs running on multiple I-stream engines (CPUs).

  In addition, the input/output (I/O) support in the TPF system coordinates the processing of channel programs (sequences of ESA I/O commands) on multiple engines in the channel subsystem. However, unless a distinction is made, multiprocessing refers to the coordination of programs running on multiple I-stream engines. Two forms of multiprocessing incorporated in the TPF system are called loosely coupled and tightly coupled.

The phrase *parallel processing* is useful to describe multiprogramming and multiprocessing because whatever the name, the *three Ss* cannot be overemphasized:
- Shared
- Synchronized
- Sequential.

A description of parallel processing emphasizes those moments where a program or processor must wait because a shared resource is already accessed by another program or processor. Some of the moments of synchronization are handled exclusively by the hardware, while others form part of the TPF software structure. When the software becomes involved in synchronization moments, the granules of time become longer than those taken by the hardware and the details become more important if the system is to be understood. When there is a failure to synchronize processes properly, competing processes can mutually block each other, possibly degrading the performance of the entire system (this is called *deadlock*).

There are some ideas we assume from the outset. For instance, the notion of *operations executed on a computer* is taken for granted. An operation is a rule for deriving output from a given input within a finite time; that is, when an operation is executed by a computer, it always ends in a prescribed interval. ESA instructions are examples of operations. A sequential process, or simply process, involves the execution of a set of operations in a prescribed order for the purpose of producing a result. The result of a process is not required to be produced in a prescribed interval, but a process must end. Some useful sequences of operations, however, never end; for example, a control sequence of code that accepts work on demand. A *program* is the passive form of a process.

Concepts of Parallel Processing

Various concepts have developed that help us understand the complexities of parallel processing.

- *Synchronization*: Any constraint on the order in which operations are carried out with the implication that the operations belong to different processes.
- *Concurrent Processes*: Whenever the first operation of one process is started before the last operation of some other process is completed.
- *Simultaneous Processes*: When the operations of several processes can occur during the same moment of time. Notice that simultaneous processes are concurrent processes, but the opposite is not necessarily the case.
- *Mutual Exclusion*: When, for the purpose of exclusively using a shared resource, one process blocks the further progress of other concurrent processes that require the use of the same resource.

- *Critical Region*: A set of operations within a process that invoke mutual exclusion. This implies that most of a process does not operate within a critical region.
- *Lock*: An implementation of mutual exclusion in which a shared resource is explicitly identified.
- *Mutual exclusion* is necessary when:
  - A shared resource can be used by only one process at a time and
  - Access to a shared resource must be granted to one of several concurrent processes within a finite time and
  - A process must release a shared resource within a finite time.
- *Deadlock*: A state where some processes fail to terminate because they are waiting on (shared) resources that are being held by each other. See Table 3.

Table 3. Deadlock

Time	Process X action	Process Y action
t(1)	Process acquires resource A	Process acquires resource B
t(2)	Holding A	Process requests A but is told to wait; Holding B
t(3)	Process requests B but is told to wait; Holding A	Waiting for A; Holding B
t(4)	Waiting for B; Holding A	Waiting for A; Holding B
⋮	⋮	⋮
t(infinity)	Waiting for B; Holding A	Waiting for A; Holding B

Deadlock

Ordinarily, deadlock does not occur. However, it does occur when all of the following conditions are met:

- A resource is held by only one process at a time. If processes can all have access to the resource at the same time, either there is no need to share the resource or the resource can be used by all processes — like electricity or read-only records.
- A resource can be released only by the process that acquired it.
- A process attempts to acquire (that is, hold) more than one shared resource at a time.
- A process can wait for a resource to be made available by another process. When two processes are waiting on resources held by each other, circular waiting, often referred to as *deadlock*, occurs.

When these conditions are combined, a resource can become unavailable. The standard example is of two processes, each of which requires two tape drives. When each process gets one tape drive and waits for the other, deadlock occurs. To elaborate, suppose a system has only two tape drives:

- Each process holds a tape drive. The process has exclusive use of the tape drive until it releases the drive.
- Only the process holding the tape drive can release it.
- The tape drives are acquired one-at-a-time by the processes.
- The processes are going to wait until a second tape drive becomes available.

Unless one of these processes is interrupted in some way (timed out, for instance) or ended, the two tape drives will continue to be held for a very long time.

Of course, if both processes wait indefinitely, an attribute for being a process is lost, that is, terminated, and some currently held resources are not released in a finite time. Normally, it is not practical to expect a process to include a time-out facility to ensure that shared resources are released in a finite time. However, it is expected that there is an external mechanism that can time out a process when necessary.

Deadlock can be detected and eliminated with system timeouts and process deletions. The TPF system protects itself in these and other ways from deadly embrace.

Deadlock Detection

The TPF system provides a deadlock detection routine to assist in detecting deadlock conditions. The routine is time-initiated and is activated during restart. If a deadlock condition is detected, a deadlock user exit (CLUD) is activated on the processor where the deadlocked ECB is located. Return code 8 is the default. Following are the return codes provided by the user exit:

- If the return code from the user exit is 8:
  1. The CE1SUD and CE1SUG fields of the ECB are set to X'81'.
  2. The waiting input/output block (IOB) that is associated with the deadlocked ECB is removed from the waiting queue.
  3. The post-interrupt routine in the IOB is activated.
- If the return code from the user exit is 4, the ECB is scheduled to exit with dump D9.
- If the return code from the user exit is 0, the ECB remains deadlocked.

You can use the ZECBL command to force a deadlocked ECB to exit with dump D9.

TPF, ESA/370, and ESA/390 Architecture

The ESA hardware influences both the function and structure of the TPF system. In a sense, the TPF system extends the function provided by the hardware.

The ESA Configuration

IBM Enterprise Systems Architecture/390 (ESA/390) is the next evolutionary step in the IBM System/360, IBM System/370, IBM System/370-XA, and IBM Enterprise Systems Architecture/370 (ESA/370) lines. Concepts from all of these systems that apply in particular to the TPF system are described in this configuration section.

The term *I-stream engine* rather than the term *CPU* is used to emphasize that a single central processing unit (CPU) is only one component of a set of hardware comprising an ESA configuration. An I-stream engine interprets one sequential stream of instructions at a time and, within the context of IBM ESA/390, implies IBM ESA/390 instructions.

Evolutions in IBM large processor architecture have emphasized the use of multiple I-stream engines that share main storage and at least one channel subsystem to manage I/O. A channel subsystem can be characterized as a set of access paths to I/O devices. This is all packaged together and takes the place of what, in the past, was frequently but inaccurately called *the CPU*. Figure 7 on page 25 emphasizes that main storage is shared among the CPUs (I-stream engines) and a channel subsystem.

The formal ESA architectural term for the structure shown in Figure 7 on page 25 is *configuration*, denoted in this publication as *ESA configuration*. An ESA configuration implies one or more I-stream engines, shared main storage, and at least one channel subsystem, without regard for the devices that can be attached to the channel subsystem. This means that devices can be attached to several channel subsystems where each channel subsystem belongs to a different ESA configuration. So, a device can access different main storages over access paths unique to an ESA configuration. A single main storage is the principal attribute of an ESA configuration.

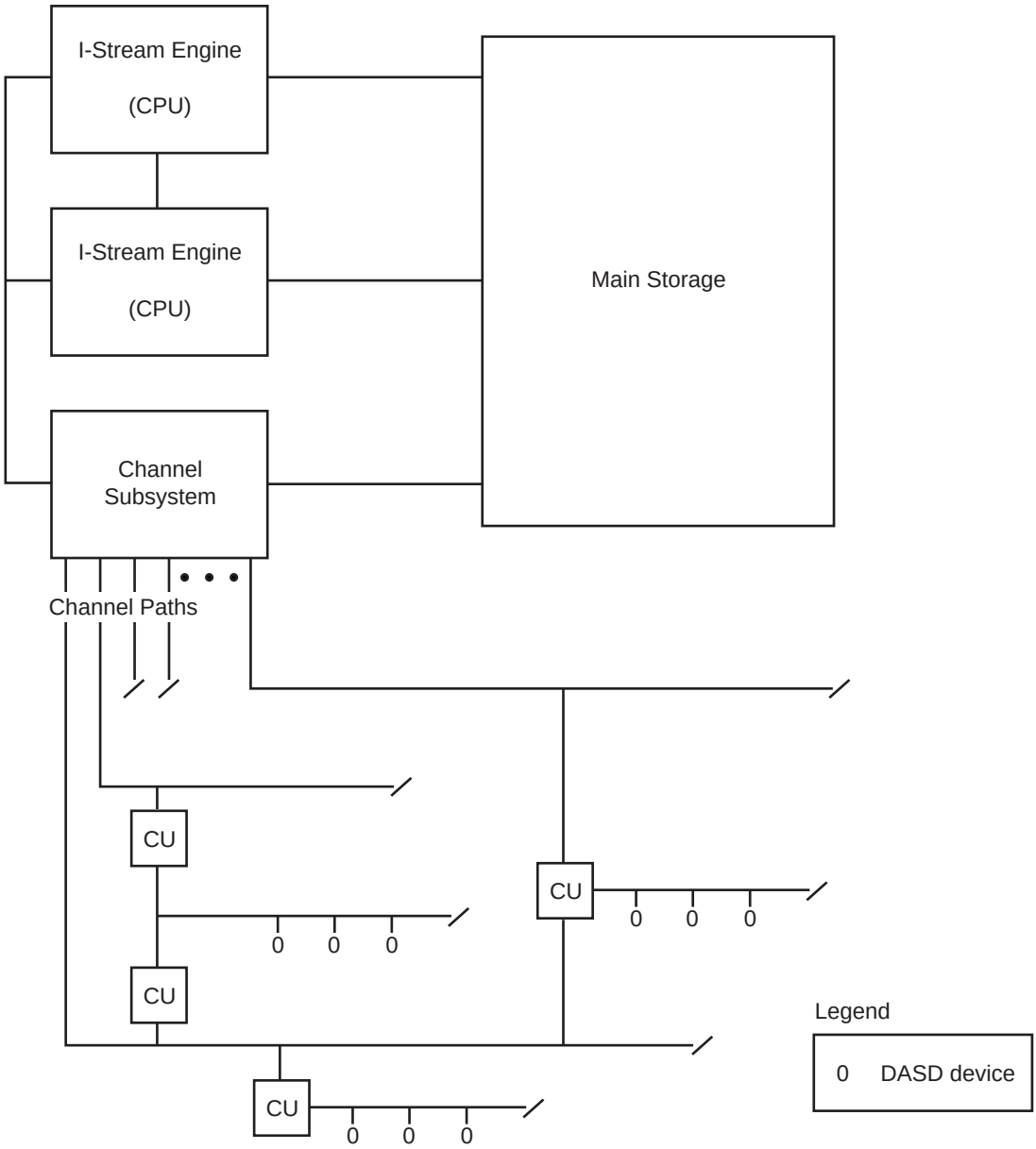


Figure 7. Logical Structure of an ESA Configuration with Two CPUs

Virtual Addressing

Main storage is viewed as a long horizontal string of bits. The string of bits is subdivided into units of 8 bits, called a byte. Each byte location in storage is identified by a unique integer starting with zero (0), called an address. Addresses are either 24-bit or 31-bit integer values.

Three basic types of addresses are recognized for addressing main storage: absolute, real, and virtual.

- An absolute address is the physical address assigned to a main storage location. No transformation is performed on an absolute address.
- A real address is identical to an absolute address with the exception of prefixing. Prefixing is used to assign a private 4 KB block of main storage (with an address range of 0 to 4095) to each I-stream engine in an ESA configuration.
- A virtual address refers to a range of addresses that can appear to be larger than the physical size of main storage. An ESA configuration implements virtual addressing through a hardware facility called *dynamic address translation* (DAT) and its associated segment and page tables.

  Each segment table points to numerous page tables and each page table points to numerous pages. A page (4096 bytes) is the smallest unit of main storage for managing blocks of virtual storage.

A virtual address, in effect, is a code that tells the TPF system how to look up the absolute address in system tables. For example, a program processes a branch instruction. This branch instruction has an address as its target. This virtual address (the target address in the branch instruction) is presented to the decoding process and the absolute address is returned. When a virtual address is used by a CPU to access main storage, it is first converted by dynamic address translation (DAT) to a real address and then by prefixing, to an absolute address.

An address space is a range of virtual addresses. The addresses are usually contiguous, but they need not be. A page is 4096 bytes and is the minimum size of an address space. A program of fewer than 4096 bytes fits into a single page. All the addresses used in a program are set up assuming the program is loaded into main storage starting at location 0. In reality, it isn't, but this assumption makes decoding the virtual address somewhat easier. If the program size increases beyond 4096 bytes (the size of a single page), another page is allocated. It doesn't matter whether the newly allocated page is adjacent to the first page or a hundred pages away from it. The TPF system decodes the addresses in the same way.

The translation of addresses is controlled by a bit in the program status word (PSW), the dynamic address translation (DAT) mode bit (DAT-mode bit). When the bit is set to 1 (assuming DAT hardware has been installed on the processor), the translation of virtual addresses to absolute addresses proceeds automatically. For programs running with dynamic address translation enabled, address references are automatically interpreted as virtual addresses during processing of the program.

The TPF system makes use of two types of address spaces provided by the ESA architecture: primary virtual address space and home virtual address space. The main architectural difference between these address spaces is the use of different segment tables for address translation.

Use of these virtual address spaces permits the TPF system to support up to 2 gigabytes (GB) of main storage and to provide a level of Entry protection for application programs. In the TPF system, primary virtual address space is called *ECB virtual memory* (EVM), and is the only view of storage available to an Entry; home virtual address space is called *system virtual memory* (SVM), and there is one SVM for each I-stream engine in an ESA configuration.

A special use of the home virtual address space occurs on system IPLs and is called the *IPL Virtual Memory* (IVM). The IVM and SVM are similar except that page and segment tables, which are defined in the IVM, are not accessible in the SVM.

Demand Paging: Virtual addressing is frequently associated with demand paging of both programs and data. In demand paging systems, some pages are loaded into main storage to help refer to their information. These pages are said to be *paged-in*. When an address does not appear in a page currently loaded in main storage (called a paging exception), pages can be *paged out* to secondary storage (usually modules to permit other pages to be *paged in*. One effect of paging is that main storage appears larger than it physically is.

Although other IBM operating systems use the dynamic address translation (DAT) facility in conjunction with demand paging, the TPF system, in fact, has ***not*** implemented demand paging. This is because all programs in the TPF system are assumed to use only a trivial amount of I-stream engine service (see the processing assumptions in “TPF Processing Assumption and Performance” on page 13). The overhead for paging out is greater than simply allowing a TPF program to complete.

Prefixing and Page 0: Within an ESA configuration, most of the main storage is shared among all the I-stream engines. An address reference used in any I-stream engine normally locates to the same absolute address. However, each I-stream engine is given a unique area of main storage, addressed as locations 0 through 4095. This area is private storage and is called *page 0*. It is accessed using a prefix register. The prefix register of each I-stream engine is loaded with an absolute address, from the full range of main storage addresses, to mark the beginning of the private 4 KB block. The hardware translates any storage reference in the range of 0–4095 to the absolute address identified in the prefix register. To ensure the uniqueness of page 0 storage, the absolute addresses loaded in the prefix register of each of the I-stream engines in an ESA configuration must be unique. Within the TPF system, the prefix registers are loaded with address values that point to the high end of main storage (see Figure 8 on page 28).

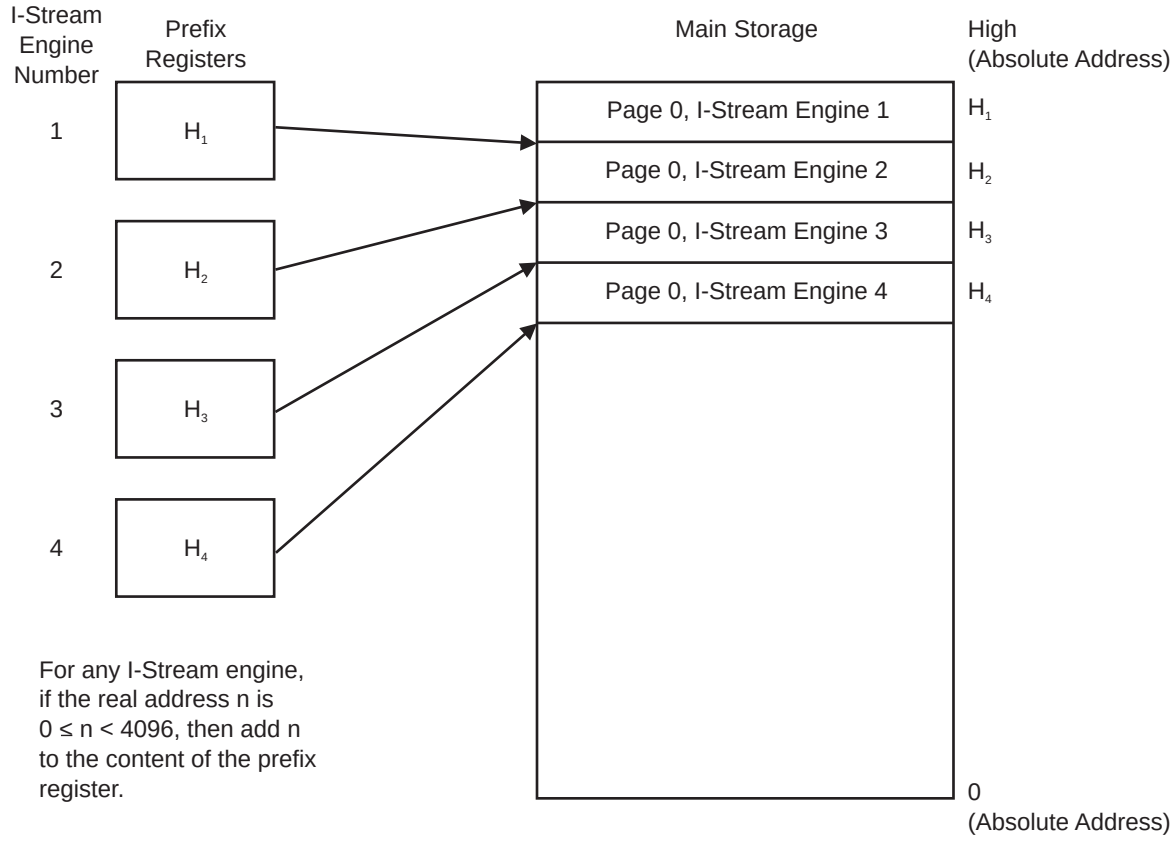


Figure 8. Page 0 Prefixing

Page 0 contains data critical to system operation. When a program running on a particular I-stream engine is interrupted, the current program status word (PSW) is saved in page 0 for the I-stream engine. Page 0 also contains indirect references to main storage. This permits the further creation of private blocks of storage for an I-stream engine. For example, within the TPF system, an I-stream engine ID (a number) is held in each page 0 of an ESA configuration. This ID is frequently used as an index into a system table, held outside of page 0, to locate a value unique to the identified I-stream engine. When such a technique is employed to find I-stream engine unique values or tables, the phrase *via a page 0 reference* is used. Page 0 is used to permit identical code, held in shared main storage, to be simultaneously executed in multiple I-stream engines within an ESA configuration.

Storage Protection

The TPF system makes use of three kinds of protection to protect main storage from destruction or misuse.

- Key-controlled protection minimizes unauthorized fetches and stores of data. A program can store data in main storage only when the storage key matches the access key. Access keys are associated with all the 4 KB blocks in main storage.

  A program's access key is associated with the main storage where the program resides. The access key consists of a 4-bit binary code followed by 3 bits further describing the access permitted. The fetch-protection bit governs whether storing alone is monitored or whether both storing and fetching are controlled. The reference bit indicates fetching and storing references. The change bit is set whenever a byte has been stored into. These storage keys are not part of addressable storage.

- Page protection controls access to main storage by protecting against unauthorized storing into main storage. However, it does not prevent fetching from page-protected memory.
- Low-address protection protects the first 512 bytes of page 0 from destruction. This area is critical during interrupt processing. One difference between low-address and key-controlled protection is that low-address protection does not prevent corruption by the channel subsystem but key-controlled protection does.

In addition to these kinds of protection provided by the ESA architecture, TPF also provides macro authorization software and address space isolation for Entries (ECBs).

Hardware Storage Arbitration

Arbitration logic, incorporated in an ESA configuration, handles the moments when more than one I-stream engine references the same main storage location during the same cycle. Only one I-stream engine gains access to the storage location, while others wait. Contention is minimized through multiple paths to main storage and by private I-stream engine storage buffers. Hardware and software synchronization resolves other main storage conflicts among I-stream engines and channel engines; for example, protection against overlaying data in I/O buffers.

Test and Set Instruction

On occasion within an ESA configuration, programs executing simultaneously in two or more I-stream engines attempt to execute or change the same area in shared main storage. A common technique for controlling access to a critical region of code is to set a bit, called a *lock indicator*, indicating the critical region of code is in use and that modifications of shared storage are being done. For example, a bit can be used to indicate that a certain data area is to be exclusively modified by only one I-stream engine at a time. Setting the bit is done prior to entering the critical region of code. Other I-streams ready to modify the same data area check the bit prior to entering the critical region. If they find the bit set, they wait until the bit is free. Unfortunately, testing and setting of the bit can take more than one machine cycle. An I-stream engine that has tested a bit and found the area it controls available can be interrupted before it can set the bit for its own use. When the interrupted I-stream engine returns from the interruption, it (perhaps erroneously) can regard the critical region as available, set the bit, and continue into the critical region. If this happens, more than one I-stream engine could use the critical region, causing severe damage.

The solution to this problem is quite important for operating systems in general. Normally, in the TPF system, the controlled data area is a system table shared among all the I-stream engines. Proper serialization of modifications to shared data is critical to the correct operation of the system.

An example showing the need for exclusive control of a shared system table is given in “Processor Lock” on page 44. For the moment, there is the additional problem of just synchronizing the bit setting among several I-stream engines.

In a multiple I-stream environment, a problem can result if more than one instruction is used to:

1. Fetch the bit
2. Check the bit setting:
   - If off (bit = 0), turn the bit on
   - If on (bit = 1), wait or do something else until the bit is turned off

Let's look at a situation that illustrates the problem: It is necessary for a program that is executing on two I-stream engines at the same time to modify a shared system table without corrupting it. Here is the program, but first note that some liberty has been taken with the instruction formats to avoid the need for introducing unnecessary coding detail.

```
BUSY: TM(0)      (Test main storage lock bit for 0)
      BZ OFF     (Branch if off)
      B  BUSY    (Wait for lock bit to become 0)
 OFF: OI(1)      (Set lock bit in main storage to 1)
      'critical' (Critical Region to modify the shared table)
      NI(0)      (Reset the lock bit to 0)
      'exit'
```

Table 4 shows the relationship of instruction execution within each I-stream engine to time and the setting of the lock indicator at each step. Remember that each instruction requires I-stream engine cycles to gain access to shared main storage under the arbitration previously described. For this timing sequence, assume that the program is executed on two I-streams separated by one cycle (one *tick* of the I-stream engine clock). The granularity of the test under mask/branch/and-immediate instruction sequence in Table 4 allows the lock indicator to be defeated.

Table 4. Timing Sequence

Time	I-stream A	Lock Indicator (after execution)	I-stream B
t(1)	TM(0)	0	"delayed" by arbitration
t(2)	BZ OFF	0	TM(0)
t(3)	NI(0)	1	BZ OFF
t(4)	enter "critical"	1	NI(1)
t(5)	——	1	enter "critical"

This problem can be solved by using one of the following instructions:

- Test and set (TS)
- Compare and swap (CS)
- Compare double and swap (CDS).

In essence, all of these instructions permit a field to be reliably interrogated and modified in a multiple I-stream engine environment. *Test and set* operates on a bit, *compare and swap* operates on a 32-bit field, and *compare double and swap* operates on a 64-bit field. Their commonality is that they serialize prior to operating.

Serialization is the process of prioritizing requests that are made at exactly the same instant, causing the requests to occur one after the other. This ensures that main storage is not going to be changed by two different I-stream engines at the same time.

For example, if programs running in multiple I-stream engines simultaneously issue a test and set instruction to check an indicator that is 0, only one I-stream engine is informed that the bit is 0 when the instruction started. All other I-stream engines are shown a 1. Furthermore, when all the I-stream engines finish the execution of the Test and Set instruction, the bit is set to 1 in main storage. Unless you intend to modify some critical system code, learning the details of these instructions is unnecessary. The general idea presented here is necessary to understand some of the system locking procedures, control blocks, tables, and macros.

These instructions, test and set, compare and swap, and compare double and swap, are sometimes called *interlocking instructions* because they can result in a coordinated delay of several I-stream engines.

The TPF system favors the use of the test and set (TS) instruction because frequently, lock indicators are bits that are set to control access to critical regions of system code. The test and set (TS) instruction requires fewer registers than the compare and swap (CS) instruction and requires less execution time because only a single byte needs to be set.

Statistically speaking, in most cases, a shared table is needed by only one I-stream engine at any point in time. If an attempt to access a shared table that is locked occurs, then within the TPF system, the program in the other I-stream engine generally enters a loop. The loop consists of testing the indicator for the right to access the shared table. Such a loop is called a *spin lock*. This loop, a software granule of time for synchronization, takes longer than the time the hardware takes to synchronize the bit setting. However, critical regions of TPF system code that update a shared table are usually only a few instructions. Very little time is wasted with spinning, because the spinning is seldom invoked, and if spinning is invoked, it lasts for only a few instructions.

CPU Serialization

An I-stream engine processes instructions one at a time. The processing of one instruction precedes the processing of the following instruction in the order in which the instructions appear in storage. This is called the *conceptual sequence*. Moreover, interruptions can take place between and within instructions.

During actual operation, instructions are broken down into smaller units. Their processing consists of a series of discrete steps. Depending on the instruction, operands can be fetched and processed in a piecemeal fashion, and some delay can occur between the fetching of operands and the storing of results. Within a given I-stream engine, access to shared main storage may not be in the same sequence implied by the conceptual sequence. This is related to instruction prefetching and the way the ESA hardware overlaps storage references in the control of the special private buffers, called *caches*. A serialization operation consists of completing all conceptually previous shared main storage accesses by an I-stream engine, as observed by other I-stream engines and by channel programs, before proceeding with the conceptually subsequent main storage accesses. All interruptions and the execution of certain instructions cause a serialization of CPU operations.

The operations of a conceptual sequence of code can be out of synchronization with its caches. So, there can be some delay in placing results in the shared main storage. The delay has no time limit and does not affect the sequence in which results are placed in storage. That is, the conceptual sequence is the actual sequence observed by other I-stream engines and the channel subsystem. However, the store instructions to shared main storage are completed only as a result of a serialization operation and before an I-stream engine enters the stopped state.

In a tightly coupled multiprocessing environment, operating system design makes sure that deadlock does not occur between I-stream engines. For instance, two or more I-stream engines may depend upon each other to issue a serialization operation to force an update of shared main storage. Fortunately, these details are handled by the software of the TPF system.

Keep in mind that the test and set instruction enables the system to enter the critical region for just one process. The test and set instruction is executed outside of the critical region, in multiple I-stream engines. Serialization is under the control of a single I-stream engine in contrast to the interlock instructions where two or more I-streams are called.

The Channel Subsystem

A channel subsystem manages the flow of data and I/O commands to an appropriate control unit which, in turn, controls I/O devices. The ESA architecture distinguishes between commands and instructions. *Command* refers to an I/O operation performed by a channel subsystem and *instruction* implies a non-I/O operation performed by an I-stream engine, with the exception of those instructions used to communicate with the channel subsystem itself. For example, a start subchannel (SSCH) instruction is used by an I-stream engine to pass a channel program, which is a sequence of channel command words, to the channel subsystem. Although a channel subsystem is itself a multiprocessing complex, most of these details can be ignored in this publication without distorting too much of the TPF system structure.

External Lock Facility (XLF): The TPF system uses an external lock facility (XLF) to maintain data integrity for shared modules in a loosely coupled complex. XLF must be connected to and shared by all ESA configurations in the loosely coupled complex. There are several types of XLFs:

- Limited lock facility (LLF)

  LLF is a hardware feature required for module CUs shared among multiple CPCs in a loosely coupled complex. The hardware feature of an XLF includes storage in the control unit (CU).
- Concurrency filter lock facility (CFLF)

  CFLF is the TPF support for the multi-path lock facility (MPLF). CFLF and MPLF are companion features to the 3990 Multi-Path Record Cache hardware feature. The hardware feature of an XLF includes storage in the control unit (CU).
- CF record lock support

  CF record lock support provides an option of using one or more CFs as XLFs.

XLFs control (serialize) access to records in the database in a loosely coupled complex. A TPF system protocol, built upon these facilities, prevents other ESA configurations from accessing a record until the lock identity is removed from the lock table.

When a module I/O request is serviced by the TPF system, an I-stream engine sends the identifier of the requested record to the XLF. A lock table in the storage of the XLF holds the identifier of all the TPF records currently being modified (and therefore, held) by any one of the ESA configurations in the loosely coupled complex. If a lock identifier is held in the table, the XLF does not permit another request to place the same lock identifier in the table. Access to such a locked record by other ESA configurations is blocked until the lock identifier is removed from the table. The data used as a lock identifier differs based on the type of XLF being used. Additional detail of the XLF is contained in Data Organization, which describes the concept of record holding.

Note: XLF is a generic term in the TPF vernacular that is used to describe the hardware facilities and associated programming used in support of shared modules in a loosely coupled complex. XLF does not refer to any particular hardware facility.

Interrupt Processing

The interrupt mechanism is the means for coordinating multiprogramming between an I-stream engine and the engines of a channel subsystem. An interrupt is a hardware enforced transfer of control within an I-stream engine. An interruption usually takes place after an instruction is finished and before interpretation of the next instruction is started. The logic built into the ESA architecture is sufficient to preserve the information necessary to return to the interrupted point of departure. Further, interrupts of the same kind are inhibited generally by the TPF system, at least long enough to preserve the state of the I-stream engine and to save control information and data. Ultimately, return is made to the interrupted code without loss of data. Classes of interrupts inhibited in an I-stream engine do not prevent interrupt generating signals to be set in the device controllers and devices. These signals are essentially *stacked* within the channel subsystem which presents the signals to any I-stream engine that is willing to accept the interruption.

A program status word (PSW) includes the instruction address and other information used to control instruction sequencing and to determine the state of the I-stream engine. A PSW also includes the bits used to inhibit or permit interrupts. In addition to the current PSW, which is the PSW in control of an I-stream engine, there are PSWs associated with each class of interrupts. There are six classes of interrupts possible:

- External
- Machine check
- I/O
- Program
- Restart
- Supervisor call (SVC).

Each class of interrupts is assigned an old and a new PSW. The old and new PSWs are held in the Page 0 for the I-stream engine.

When an interrupt occurs, the current PSW is stored into the old PSW for the class of interrupt, and the new PSW for the class of interrupt is loaded into the current PSW.

Hardware Processing of Concurrent Interrupts: Consider the example of processing two concurrent interrupts occurring in an I-stream engine, one is an input/output (I/O) interrupt and the other an external (EXT) interrupt. Assume the interrupted program is at location NSI-1, where NSI means the *next sequential instruction*.

The hardware interrupt processing accomplished by the ESA hardware is reviewed to emphasize the sequential processing done on the PSWs when concurrent interrupt forcing signals are presented to an I-stream engine.

The time sequence for the processing that takes place is given in Figure 9 on page 34, where:

IADDR	= Address of I/O Interrupt Handler
EADDR	= Address of External Interrupt Handler
NSI	= Next Sequential Instruction
___	= Not relevant

The arrows show data movement.

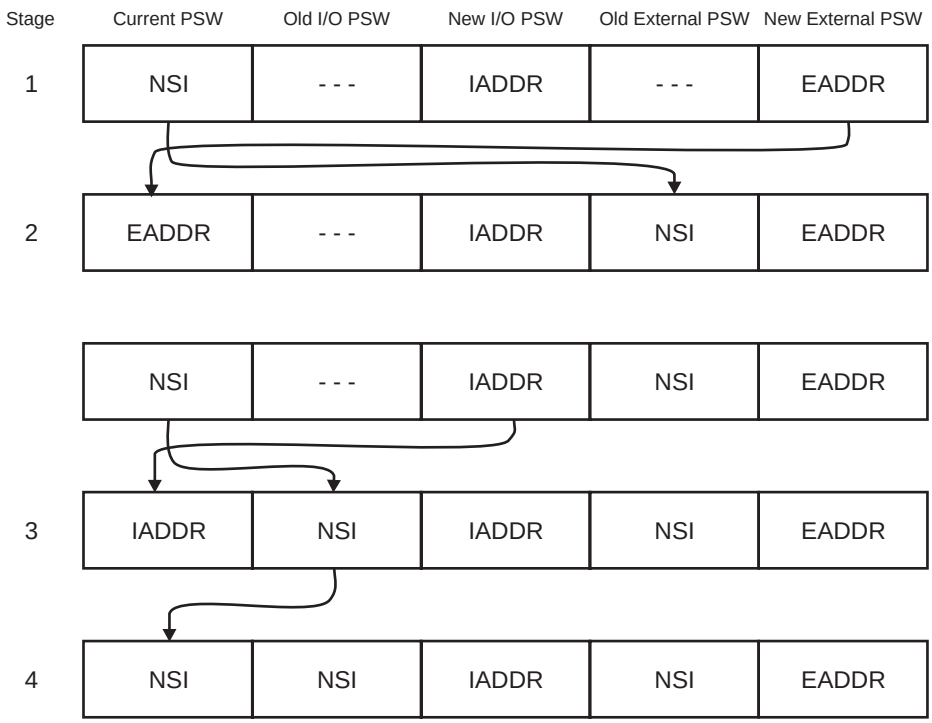


Figure 9. Concurrent Interrupts

Software Processing of Concurrent Interrupts:

1. When the concurrent interrupts occur, assume that the PSWs are set as shown at stage 1 in Figure 9.
2. When the hardware performs PSW swapping, the current PSW is set to pass control to location EADDR, which is the TPF program called the *external interrupt handler*. At this point, both I/O and external interrupts are disabled. (Stage 2 in Figure 9.)

   The concurrent I/O interrupt is stacked by the hardware because I/O interrupts are disabled in the external new PSW when loaded, preventing the I/O interrupt from being honored at this time.
3. When the external interrupt handler completes, the *old external* PSW is loaded as the *current PSW*. At this time, the pending I/O interrupt stacked by the hardware is honored because the PSW associated with NSI has I/O interrupts enabled. Control is passed to location IADDR by the PSW swapping mechanism. (Stage 3 in Figure 9.)
4. When the I/O interrupt handler completes, the *old* I/O PSW is loaded into the current PSW, which returns control to location NSI in the interrupted program with all interrupts enabled. (Stage 4 in Figure 9.)

Clearly, the external interrupt receives higher priority than the I/O interrupt. The TPF interrupt processing code depends upon the hardware for *stacking* unprocessed interrupts, while the TPF system must ensure that data is not lost. This is done through judicious setting of the mask bits. A disabled (masked off) interruption condition is retained in the hardware, and when software processing of the interruption event has completed, re-enablement occurs to permit any further

interruption to take place. A load PSW instruction may load the NSI (next sequential instruction) of the interrupted program as well as accomplish interruption enablement at the same time.

Significance of Interrupt Handling by the TPF System: Interrupt handling emphasizes the importance that the design of the system places on meeting the demands of the current program as quickly as possible.

When an application program is interrupted, the very same program regains control immediately after the TPF system has serviced the interrupt. This is always true, except when a supervisor call (SVC) instruction is issued that specifically requests the relinquishing of control. In other IBM operating systems, the dispatching mechanism is normally called after an interrupt occurs; therefore the interrupted program may not get control back immediately after the interrupt is processed. But this is not so in the TPF system where the interrupted program usually receives control again after the interrupt. This may seem like a small detail, but it represents a fundamental difference between the TPF system and other operating systems.

Another difference between the TPF system and other IBM operating systems is the way in which an I-stream accepts a non-module I/O interrupt. Any I-stream engine in an ESA configuration is capable of accepting an I/O interrupt. This means that an I-stream engine accepting an I/O interrupt is not necessarily the same I-stream engine that issued the I/O operation causing the interrupt. However, in the TPF system, a non-module I/O interrupt is accepted only by the same I-stream engine that started the I/O operation. Furthermore, in the TPF system, not all I-stream engines are permitted to start non-module-related I/O operations. Applications running on any I-stream engine can issue the TPF macros related to I/O requests. However, a TPF I/O macro request from any I-stream engine can be moved, if necessary, to an I-stream engine that services I/O. *Move*, in this case, means that the I/O request is ultimately referred to by the registers and PSWs of the servicing I-stream engine. The process of moving work among I-stream engines is described in “Action on the Cross List (Switching I-Stream Engines)” on page 84.

System States: Problem and Supervisor: The distinction between problem state and supervisor state gives the TPF system the ability to control the execution of certain instructions that are critical to the operation of the system. These states are controlled by a bit in the PSW. Code that runs in supervisor state is permitted to execute privileged instructions. Within the TPF system, the privileged instruction set system mask (SSM) instruction is important for control of the system state.

Authorization and Supervisor State: Entering supervisor state and executing certain macros requires special authorization for a program in the TPF system. Each program has privilege class characteristics associated with it before processing. If the program is not authorized (that is, is not privileged) to process a particular class of macro and it tries to process one, an error is reported and the program is ended.

This prevents an unauthorized program from issuing macros that are intended solely for system control. Without the authorization facility, the TPF system would be vulnerable to corruption.

Interrupts and System State: This discussion further illustrates the TPF system’s emphasis on meeting the demands of the current program as quickly as possible. At this point, it is necessary to distinguish between a hardware interrupt and a

software interrupt. Program, restart, I/O, machine check, and external interrupts are classified as hardware interrupts. An SVC interrupt is classified as a software interrupt.

When a hardware or software interrupt arrives while the TPF system is in problem state, as part of the hardware reaction to the interrupt, the system state changes to supervisor state, the current processing environment is saved (PSW swapping), and control is transferred to the designated TPF interrupt handler. The TPF interrupt handlers run with interrupts disabled to prevent the TPF system from falling into an infinite loop that could occur by processing subsequent interrupts. The disabling of interrupts does not degrade the TPF system because the interrupt handlers are deliberately designed as only short sequences of code.

For a software interrupt, the interrupt handler (that is, the macro decoder) identifies the action to be taken as a result of the interrupt, re-enables interrupts, and transfers control to a system program (a macro service routine) to perform the action. The system program, still in supervisor state, processes the action related to the interrupt, puts the system in problem state, and returns control to the program that was interrupted. When another software interrupt arrives, the process repeats itself.

For a hardware interrupt, the interrupt handler identifies the action to be taken as a result of the interrupt, queues the remaining processing for subsequent processing, re-enables interrupts, puts the TPF system into problem state, and returns control to the program that was interrupted. When another hardware interrupt occurs, the process repeats itself.

The various system programs that process in supervisor state mask interrupts depending on their types of processing so that when hardware interrupts do occur during their processing, the interrupts are stacked and subsequently processed once interrupts are unmasked.

The privileged instructions *set system mask* (SSM) and *load PSW* (LPSW) are used by the TPF system programs to change the system state.

SVC Interrupt and System State: The supervisor call (SVC) interrupt represents a deliberate request for a system service by an application program. For example, some of the system services that are called by SVCs are:

- Read a record from a module (the FINDC macro)
- Write a record to a module (the FILEC macro)
- Create another Entry (the CREMC macro).

SVC interrupts cannot be disabled; however, the only way such an interrupt occurs is through the processing of an SVC instruction. The TPF system maintains control by effectively processing only one SVC for each I-stream engine at a time.

To repeat, although there are important exceptions in the TPF system, the same program that issues the SVC (which causes an SVC interrupt) regains control immediately after the TPF system has serviced the request. This is a fundamental difference between the TPF system and most other operating systems.

Summary of Interrupts: The acceptance of an interrupt within an I-stream engine is controlled by a PSW and related control registers. Interrupts and PSWs are essential for controlling I/O operations between the channel subsystem engines and an I-stream engine and for providing system services to applications.

Central Processing Complex (CPC)

Choosing the right terminology is, to some extent, a packaging phenomenon of a system design. For example, several central processing complexes (CPCs), all sharing modules, are supported by the TPF system. At some TPF installations, this represents a central processing site. However, as different forms of local and wide area communication interconnections become universal, the use of the term *central* must be used with caution. Some TPF installations, for instance, do not exist entirely in the same building. We still use the term central processing complex (CPC) to denote an ESA configuration attached through a channel subsystem to a set of devices or other ESA configurations. Figure 10 on page 37 shows interconnected loosely coupled complexes where each CPC is an ESA configuration that can include multiple I-stream engines and a set of private devices such as tapes and modules. CPC is used to emphasize attachments that, architecturally, are external to an ESA configuration.

CPCs that share the module configuration as in Figure 11 on page 38 are connected through channel-to-channel (CTC) support.

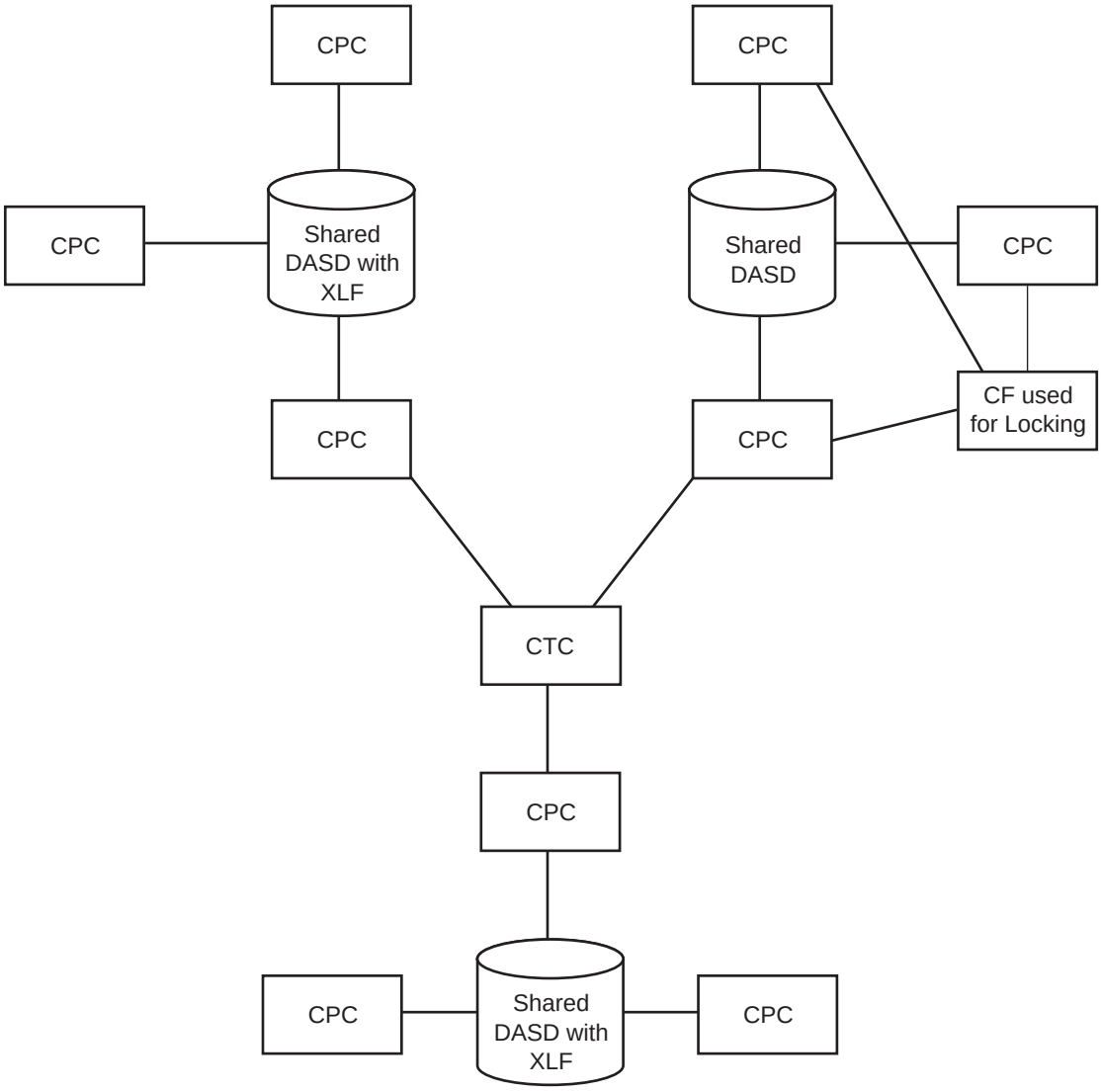


Figure 10. Interconnected Loosely Coupled Complexes

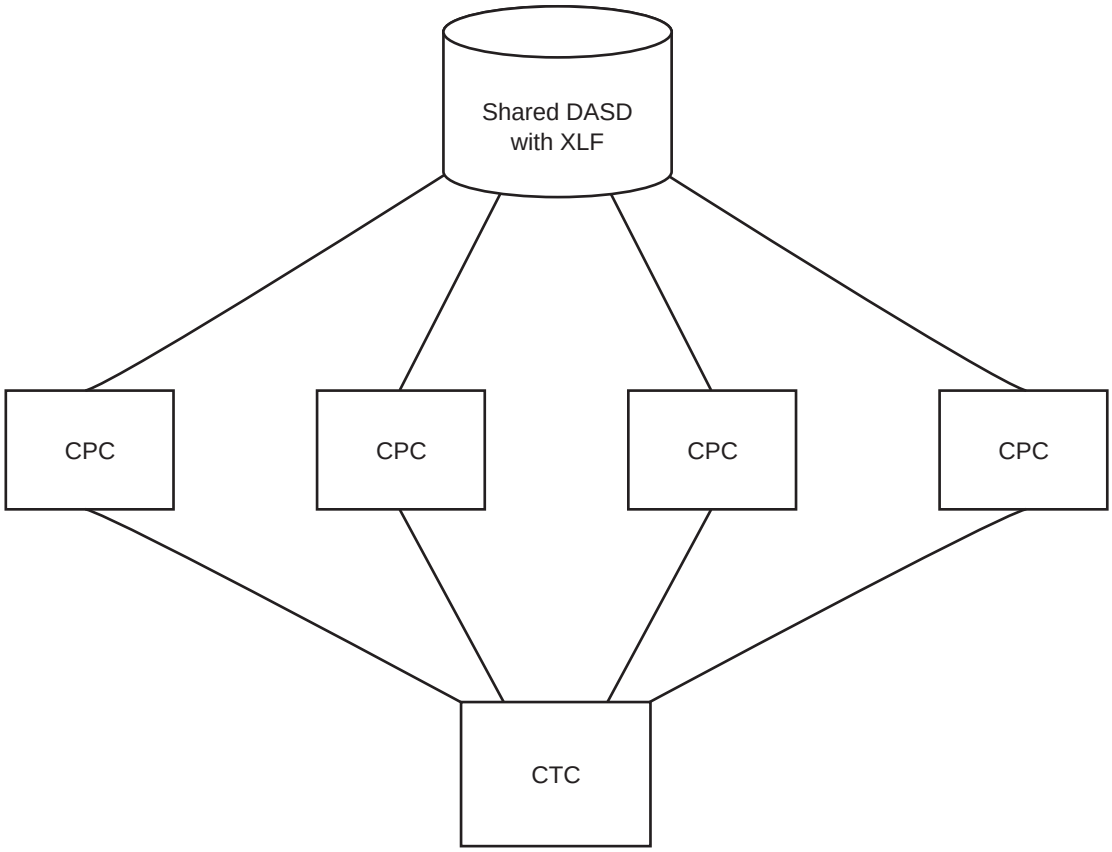


Figure 11. Loosely Coupled Complex

Interprocessor Communication

Both loosely coupled and tightly coupled multiprocessing require mechanisms for interprocessor communication to coordinate the work distributed among a set of cooperating processors in a variety of arrangements.

Essentially, there are several forms of interprocessor communication:

- There is communication that takes place across channels using the Multi-Processor Interconnect Facility (MPIF) feature that is used to link central processing complexes (CPCs) in a loosely coupled complex.

  Interprocessor communication is associated with a mechanism for sending messages among ESA configurations in a loosely coupled complex, where the content of the message does not necessarily imply that a lock is called.
- There is also communication between programs running in different systems that are controlled by different operating systems using the TPF Application Requester (TPFAR) feature.

  Using the TPF Application Requester (TPFAR) feature, a request for data retrieval from an IBM DATABASE 2 (DB2) database can be sent from a TPF system to an IBM MVS or IBM VM system.

TPF System Program Structures

A distinction between application and system programs is helpful. Also, two different program structures called *reentrant* and *serially reusable* identify some of the synchronization issues inherent in parallel processing, whether a program is classified as an application program or as a system program.

Application and System Programs

A functional distinction between application and system programs is:

- Application programs are used to interpret the semantic content of messages entered by the end users of the TPF system. An end user is, for example, an airline reservations agent obtaining a request from a customer to reserve space on a flight.
- System programs provide system services that shield application programs from many of the details necessary for sharing system resources. The system services, in many cases, are provided in the form of macros for application program use and which, loosely, form a language. For example, to read a record from modules, the FINDC macro is an SVC I/O request to read (get) a record.

Reentrant Programs

Reentrant programs become an important ingredient within a system that incorporates a high level of parallel processing. Reentrant programs allow a single copy of a program or routine to be used concurrently by two or more processes.

In this context, there is a significant distinction to be made between the terms *program* and *process* because a single reentrant program is designed to handle multiple processes. Recall, from the introduction in this chapter, that a program is characterized as the passive representation of a process. Now it is useful to think of multiple processes as *animations* of a single program. The reentrant program accommodates multiple processes through switching execution on a single I-stream engine as well as simultaneous execution on multiple I-stream engines. A distinction between a program and a process is not always necessary; for example, the phrase “an application program is delayed” implies a running program in the service of some process.

Reentrant programs conserve main storage space, eliminate some file accesses for fetching additional copies of a program to service additional processes, and permit the same copy of a program to be executed simultaneously in multiple I-stream engines. Be assured, however, that multiprogramming and multiprocessing can be implemented without using reentrant programs by consuming more main storage space, making additional accesses to external storage, and synchronizing processor execution.

The TPF system requires that ISO-C programs be reentrant, which means that the RENT compile-time option must be specified when you compile segments that contain writable static. There is a performance overhead for writable static, however, particularly when calling library functions. Consider designing ISO-C libraries and applications that do not contain writable static data. An application that does not contain writable static data is considered to be naturally reentrant and does not need to be compiled with the RENT option. This avoids the overhead associated with writable static.

TPF Entry

Process is an abstraction of several TPF constructs and, in particular, of one called an Entry. An Entry is created to do application processing. In particular, an Entry is

used to accept an input message and to produce a response (output) message, where the response can be preceded by updates to the database. The input message comes from an end user (for example, a reservations agent) and anticipates a corresponding response (for example, printing a ticket or a message that displays a flight schedule). Normally, various reentrant programs, called program segments, are run sequentially on behalf of the Entry. The concrete description of an Entry associates the input message with control blocks and related pointers. A more comprehensive description of an Entry is delayed until additional TPF system structure is introduced in TPF System Structural Characteristics. Nevertheless, an overview of the control block structure in support of TPF processes, called Entries, is shown in Figure 12 on page 41. This is an abstraction of the details and relies upon your assumed previous computing experience.

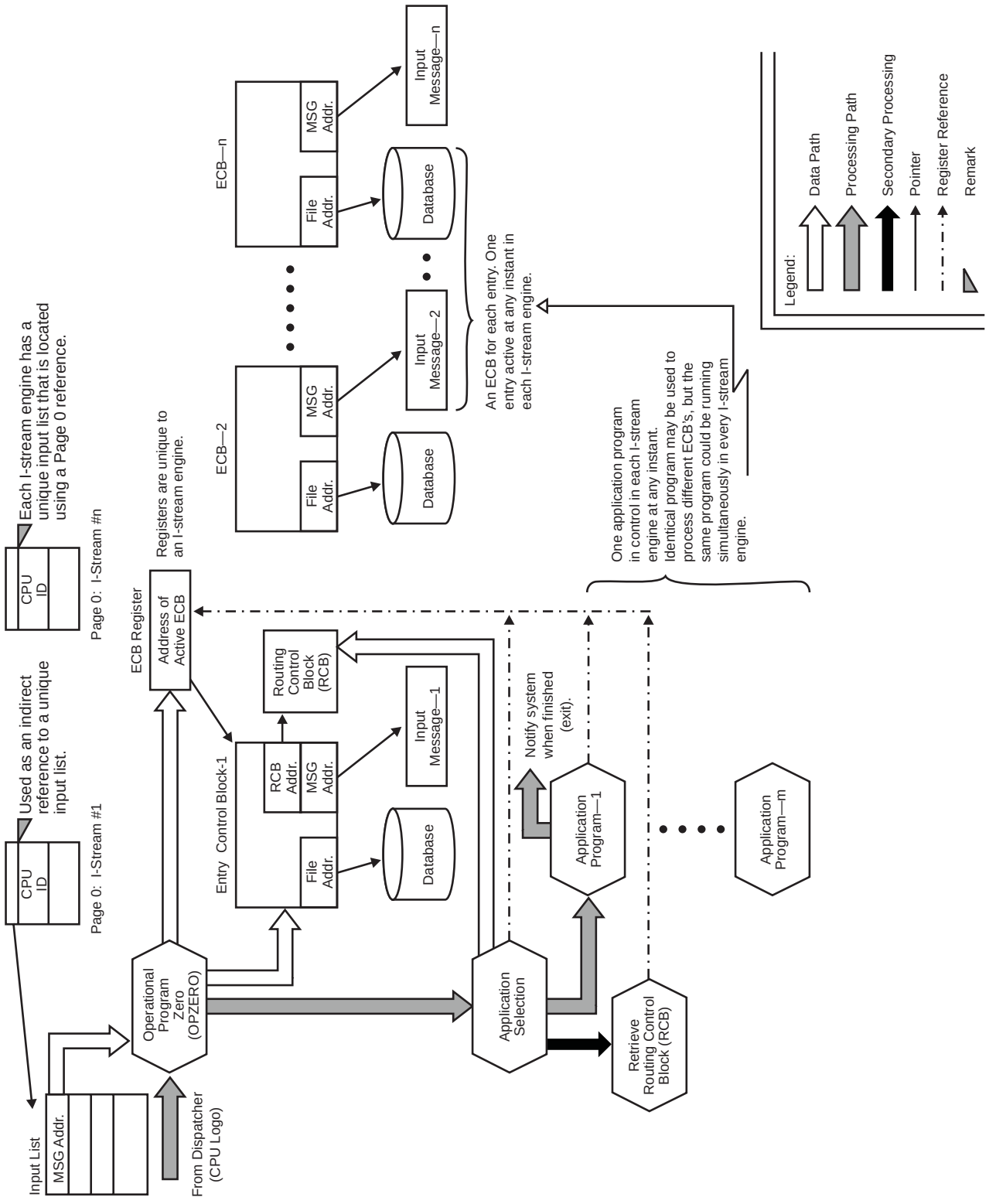


Figure 12. TPF Process Structure

The notion of reentrancy is combined with the notion of an Entry in the following example: The same copy of a TPF application program can be used to update an airline seat inventory on behalf of many reservations agents; each agent enters messages needed to request seat reservations for different customers. The unique

messages from each of several agents and the implied system actions represent several processes (Entries, in this case). The same program can be servicing several Entries on the same I-stream engine through multiprogramming and simultaneously processing on multiple I-stream engines through multiprocessing. Therefore, Entry unique data must be held in private storage areas (which in the TPF system are called the *ECB work areas*). A reentrant program indirectly references unique input and output data with private I-stream engine registers that point to the data. The values in the registers must be reset for each invocation of the program. This ensures that information used and generated as a result of processing separate Entries remains independent of the shared code invoked to do the processing. This permits the program to be reentered on behalf of additional Entries before the program computes the result of previous Entries.

A Problem of System Evolution

In the early versions of the TPF system, a primitive form of reentrant programming was practiced. Before multiple I-stream engines, it was not necessary to coordinate the updating of system tables because in a uniprocessor environment, there is only one I-stream engine.

This violation of the reentrant attribute in a multiple I-stream engine environment can be explained by considering that if the sequence of code that updates an entry in a system table is simultaneously executed in more than one I-stream engine without incorporating critical region protection, the entry can be inaccurately modified. Even though the hardware arbitrates simultaneous access to the same storage location by two different I-stream engines, this is not sufficient to ensure that any two I-stream engine executions are far enough separated in time to ensure that the second access to the table entry sees the result of the first I-stream engine's execution. This could result in losing one of the updates.

This problem is mentioned here to shed some insight on the subtleties of evolving the TPF system into a multiprocessing environment.

Serially Reusable Programs

A sequence of code that is guaranteed to run to completion before being invoked to accept another input is called *serially reusable code*. In earlier versions of the TPF system, serially reusable code was used within reentrant programs to update shared data. This was reasonable because in a uniprocessing environment, a sequence of code that updates shared data is serially reusable within a single I-stream engine. However, if the code is to remain serially reusable in a multiple I-stream engine environment, then the code can be used by only one I-stream engine at a time. Serially reusable code is a critical region in disguise.

In early versions of the TPF system, the system programs were designed on the assumption that all application programs were serviced by a set of system programs that ran on a single I-stream engine (CPU), required very little processing per request, used no I/O, and ran to completion before servicing another application request. In other words, it was assumed that system programs were serially reusable.

Many application programs are reentrant, but not all. Some refer to a shared area of main storage called the *global area*. The interval of time in which an Entry modifies the global area is not critical in a uniprocessor environment; the attribute of being serially reusable during modifications to the global area is tacitly assumed. This critical region for applications causes problems in a multiprocessing environment. The use of the global area in a multiprocessing environment affects

application program interfaces and is related to some rather complex structures for incorporating references to the global area into the application programs.

A serially reusable sequence of code may, in a uniprocessor environment, build data within a program area that is not private or indirectly referenced. However, this same code occasionally fails in a multiprocessing environment when the code is executed simultaneously on more than one I-stream engine. This is caused by updating values in a shared area that can be corrupted by the conflicting demands of separate processes in a multiple I-stream engine environment. In reality, within a multiple I-stream engine environment, a serially reusable program that is designed to run in a uniprocessor environment is no longer guaranteed to run to completion before being reused.

Several techniques apply to both system and application programs and are used within the TPF system to overcome the assumptions of serial reusability in the past:

- Restructure code sequences to be reentrant instead of serially reusable
- Allow the program to run on all I-stream engines within an ESA configuration, but restrict execution of critical regions to one I-stream engine at a time

  This means that the critical regions of existing code must be identified and controlled by the mechanisms that restrict their execution to one I-stream engine at a time.
- Restrict the use of serially reusable code to a single I-stream engine (within an ESA configuration)

  This means that within an ESA configuration of multiple I-stream engines, the program can only execute in one I-stream engine. The TPF system provides a load balancing mechanism to do this, or the user installation can design its own.
- Allocate a copy of serially reusable programs to each I-stream engine within an ESA configuration.

  This leads to various forms of interprocessor communication.

The *best* technique would be a completely new system design and implementation. However, this is an option constrained by the past: preserving application interfaces for loyal customers.

Multiprogramming Defined

Multiprogramming is defined within the context of the use of one I-stream engine because a single I-stream engine can only process a stream of instructions for one program at a time. When several program instruction streams are shared in a single I-stream engine and a dispatching mechanism exists to switch among the programs, the environment is said to be *multiprogrammed*.

The TPF system is designed on the assumption that the processing required for a single message places relatively heavy demands on database access and only a little demand on I-stream processing cycles. This assumption identifies a fundamental principle of multiprogramming in the TPF system: *To keep an I-stream engine busy when programs are delayed by I/O operations*. Multiprogramming is useful when an application program is delayed due to I/O requests, and the I-stream engine is more efficiently utilized if another program can be processed by the I-stream engine during that delay. Multiprogramming makes sense because a channel subsystem represents independent processors sharing storage with I-stream engines. So, in reality, multiprogramming on an I-stream engine within the TPF environment is done in support of multiprocessing, where a channel subsystem and a single I-stream engine represent the multiple processors. Multiprogramming is

also effective when a program must wait for the use of the services of an I-stream engine that is different than the I-stream engine on which the program is running.

There are other reasons for multiprogramming, some of which do not exist in the TPF system. For example, computationally intensive programs are not tolerated in the TPF system where most Entries complete in less than one-half second. However, multiprogramming is used in other systems to force computationally intensive programs to give up control to higher priority work.

TPF System Tightly Coupled Multiprocessing

Tightly coupled multiprocessing means more than one I-stream engine within an ESA configuration is available to process programs held in main storage that is shared among the I-stream engines. This requires the synchronization of multiple I-stream engines simultaneously that are processing sequential programs, with the common goal of increasing the number of messages that can be processed. When the system programs share data, also held in the shared main storage, the synchronization is accomplished with a TPF system mechanism called a *processor lock*. Locks, in general, are the implementation of the concept of mutual exclusion.

Processor Lock

A processor lock is used to permit system programs, executing in multiple I-stream engines in an ESA configuration, to modify system tables held in the shared main storage. In general, access to a shared system table occurs within the framework of other processing that does not require locking.

A processor lock is intended for the exclusive use of the system programs. A processor lock uses an indicator that system programs reference before entering a critical region where programs executing in any I-stream engine can modify shared variables. If a processor lock indicator is set, the program checking the lock indicator must wait until the lock indicator is released. The TPF system normally uses the test and set (TS) instruction to force an interlock across multiple I-stream engines in order to set the lock indicator. In the TPF system, the activities of testing, setting, resetting, and waiting (or spinning) are all considered to be part of a processor lock.

The record hold table is an example of a system table where processor locking is performed. The record hold table is a shared table accessed by system programs from multiple I-stream engines (within an ESA configuration) in the course of servicing an I/O request. Placing data into the record hold table is done by the critical region of a system program running on a single I-stream engine because the structure of the code is serially reusable. Refer to Figure 13 on page 45 while reading the following description.

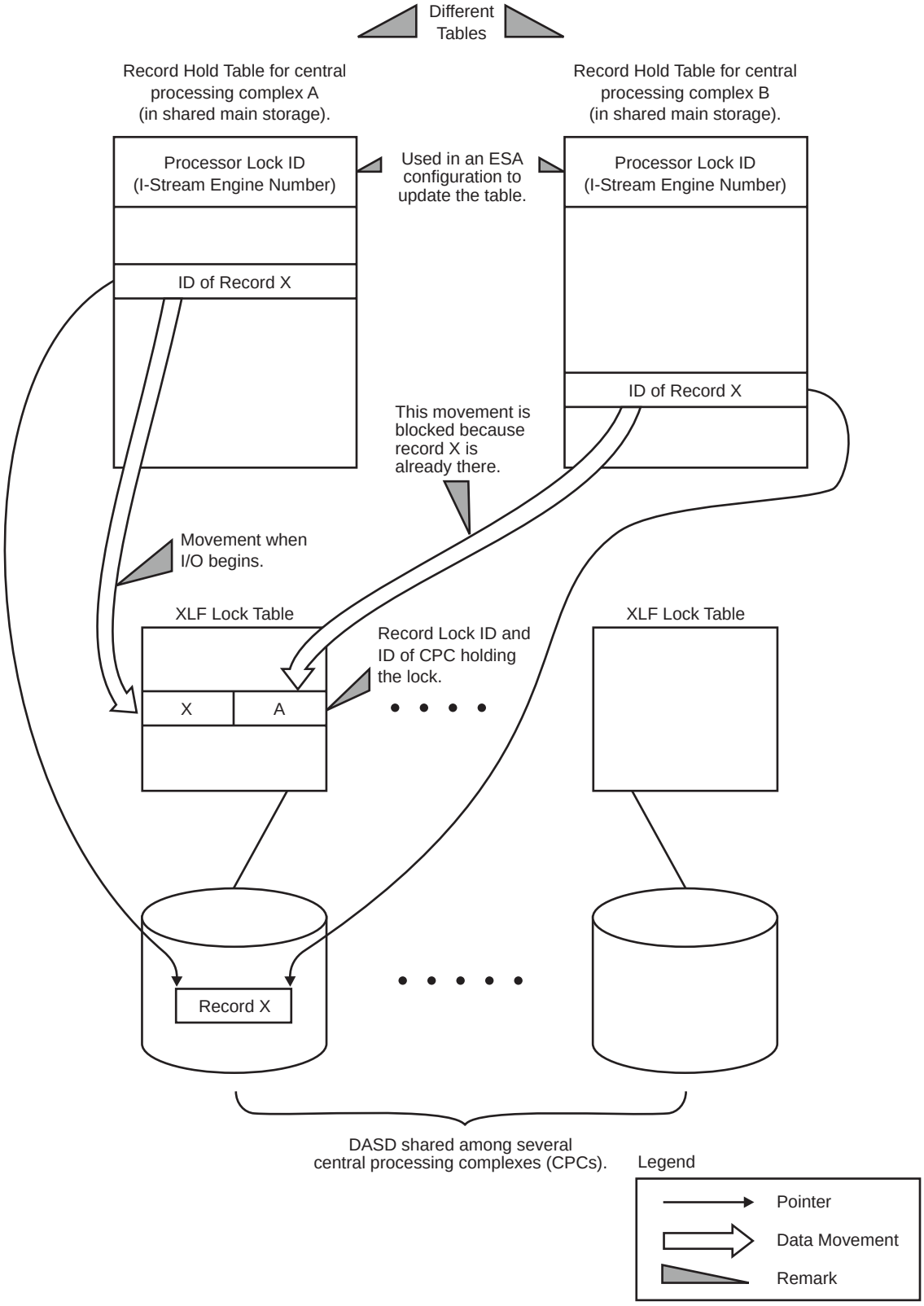


Figure 13. Relationship of Locks

The items in the record hold table identify records that are held for the exclusive use of a single process (Entry) during a record update. Through the use of a file read macro (called the FIND and HOLD macro), an Entry identifies one record that

is about to be modified. A reentrant application program that services multiple Entries has no need to even be aware of the record hold table, but only needs to know the macros used to hold and release a record.

The file address of a record placed in the record hold table is a lock indicator that shows the exclusive use of the record by one Entry within an ESA configuration (not to be confused with the processor lock on the record hold table itself). As long as the file address of a record is in the record hold table, observation of the application programming protocols prevents any other Entry from updating the record.

When an XLF is installed, the file address is given to the XLF at the time I/O commands are given to the channel subsystem to access the record. Placing a file address in the record hold table does not necessarily mean that the I/O commands are issued immediately; this depends upon the queues to the relevant module CUs. The contents of the XLF table (lock table) located in the module CU or CF may indicate that the record is already being used by some other central processing complex (CPC). The file address of a record (an entry in the lock table) locks the record for one Entry within a loosely coupled complex of several CPCs that are all attached to the same shared module. See Figure 11 on page 38 for more information.

The TPF system places the identity of records held by all Entries within an ESA configuration into the record hold table. Processor locking is called when an item is placed into the record hold table. Keep in mind that the record hold table is a single table, located in shared main storage that can be accessed by the same system code processing in all the I-stream engines within an ESA configuration. A field within the record hold table is used to hold the processor lock indicator. The processor lock prevents corruption of the record hold table. This could occur if two or more I-streams simultaneously attempt to process the single copy of system code used to place the identity of a record into the record hold table.

Three separate points of synchronization for locking shared main storage data, such as the record hold table, to note in the TPF system are:

- Use the test and set (TS) instruction to set the bit (lock indicator) that reserves access to the shared storage area. See “Test and Set Instruction” on page 29 for more information. Without this hardware assist, the processor lock indicator could be inappropriately set in the short interval between testing the indicator and setting the indicator.
- Place the data (the record lock identity, in this case) in the shared storage area. If more than one I-stream engine attempts to insert data, the TPF system makes all but one I-stream engine wait (this is not very long, but it is longer than one instruction execution).
- After the shared storage area is updated, the TPF system must release the lock indicator.

In the TPF system, the I-stream engine waiting mechanism is called a *spin lock* because any delayed I-stream engine is not doing anything but looping; that is, spinning, while waiting to access the locked resource. Fortunately, an I-stream engine is seldom delayed with a spin lock. Unfortunately, to use a section of code that accesses shared data, each I-stream engine must incur the overhead of the lock mechanism.

A side comment about the record hold table is instructive. The reason that there is a single record hold table in an ESA configuration is due to the nature of the application environment: in a TPF system, all applications are assumed to need

access to the same underlying database. Although record locking is essential, seldom does contention occur once a lock indicator is placed in the table. The small performance penalty for setting the lock is worth the large performance improvement gained by permitting concurrent accesses to a very large database. The system complexity for managing a single shared table is simpler and causes less delay than attempting to place separate tables in a private area for each of the I-streams. Separate tables would require complex logic to coordinate.

Application Locks

In the case of the example of the record hold table, there is another level of synchronization: manipulating the data in the locked record. The application programs sharing the data observe a protocol to not update a record if it is locked. The delay for gaining access to a locked record may be relatively long, and the TPF system treats this contention as an I/O delay, whereupon multiprogramming is invoked to switch to a different Entry (process).

System Program Structures

In a tightly coupled multiprocessing environment, portions of system programs that are shared in main storage are structured as reentrant code. Prior to tightly coupled multiprocessing, the TPF system had little need for reentrant system code, because within a single I-stream engine, system programs depended upon completing a service request before being invoked to process an additional request. The reentrant system code is useful because in a tightly coupled environment, a single copy of such code is simultaneously executed by more than one I-stream engine. If all system programs were to rely upon finishing before being reentered, prohibitive processor contention would occur. Most of the system programs that service application macro requests are reentrant programs.

The choice of a programming structure and related locking mechanisms is usually a compromise between complexity and performance. Restructuring tables and code can become complicated and more time consuming than development schedules permit. However, excessive locking defeats the purpose of multiprocessing.

The decision to handle non-module-related subclasses of I/O interrupts on a single I-stream engine sheds some insight on such issues. A relatively large number of heavily accessed tables are used by the system I/O programs. To restructure all these tables and associated code would be a formidable task, and to lock on each access to a table would be a severe performance penalty. Statistical modeling shows that a good compromise is to restrict the processing of a subclass of I/O interrupts to a single I-stream engine, thereby avoiding locking for a minimal performance penalty. The penalty is incurred by the need to transfer each I/O request to the I-stream engine that manages I/O from the I-stream engine that issued the request but cannot issue the hardware I/O command.

Most module-related I/O is handled by all I-streams. This requires that module-related tables must be locked. If module I/O was done on only one I-stream, that one I-stream would soon become overloaded doing only module I/O processing.

CPU Affinity

A program restricted to run on a particular I-stream engine is said to have a *CPU affinity*. A program with a CPU affinity has no need to call locks associated with critical regions if the data modified by the program is not shared by programs running on other I-stream engines. So, a program is given a CPU affinity if it accesses particular main storage tables that are not accessed by any other

programs; this eliminates the overhead of locking. Note, however, that only one copy of the program can run at any one time in the I-stream engine. Moreover, it is not appropriate to assign a CPU affinity if there are several programs that access a single main storage table. Of course, CPU affinity does not restrict a program from using locks to access data that is shared. But, incorporating a lock into an existing program represents a modification. A program designed to be serially reusable in a uniprocessor environment operates satisfactorily without modification in a tightly coupled multiprocessor environment if the program is restricted to run on a single I-stream engine, and if none of the data that it modifies is shared with programs running on other I-stream engines.

Examples of some TPF system functions that must run with a CPU affinity are non-module-related I/O interrupt processing, timer service, and command processing. Commands are called *operator messages* or *commands* in other operating systems.

The fundamental reasons for assigning a CPU affinity to a system program are:

- Locks are avoided on highly accessed tables.
- Programs designed for a uniprocessor environment can be used with minimal modification.

The trade-offs for CPU affinity are:

- Additional procedures must be created to move requests from other I-stream engines and to notify other I-stream engines about the results of a program that runs with a CPU affinity.
- Restricting a program to run on a single I-stream engine can cause queuing problems that degrade performance.

I-Stream Engine Categories

In the TPF system, for the purposes of CPU affinity, I-stream engines are assigned either as the main I-stream engine or as an application I-stream engine, (there is only one main I-stream engine in an ESA configuration). The I-stream engine (CPU) to which system programs are assigned CPU affinity is determined during initial program load (IPL). The assignments vary with the number of I-stream engines existing in the ESA configuration being IPLed. Applications (that is, Entries) can be run in any I-stream engine category. In a uniprocessor environment, all programs have CPU affinity to the only available I-stream engine, which serves both as the main I-stream engine and the application I-stream engine.

Main I-Stream Engine: There are some system programs that must be assigned an affinity to a unique I-stream engine within an ESA configuration, called the main I-stream engine. This I-stream engine is the one that is IPLed from the functional console when the TPF system is first being loaded. The timer service is an example of a function that must be assigned an affinity to the main I-stream engine. The main I-stream engine can be used for running any serially reusable system program that modifies data that is not shared among the other I-stream engines in the ESA configuration. So the main points are:

- Within an ESA configuration, there is only one main I-stream engine.

  Some system programs must be assigned an affinity to the main I-stream engine.
- Applications (Entries) can run in any I-stream engine, including the main I-stream engine.

Application I-Stream Engine: An I-stream engine to which no system programs or only a restricted set is assigned CPU affinity is called an application I-stream engine. These are all the I-stream engines in an ESA configuration, if any, other than the main I-stream engine.

The Multi-Processor Interconnect Facility (MPIF) feature is a function that can be assigned CPU affinity. By design, it can be assigned to the application I-stream engine that is designated as *I-stream engine 2*.

Performance Implication

There are some performance implications resulting from the way that tightly coupled multiprocessing is implemented in the TPF system:

- The software locking facilities that are necessary for an operating system designed for multiple I-stream engines are integrated into the architecture of the TPF system. Therefore, a small overhead is incurred when running the TPF system on an ESA configuration with only one I-stream engine (that is, a uniprocessor).

  Although this performance degradation in a uniprocessor environment is somewhat dependent on the application design, it is considered to be worth this penalty because greater performance and flexibility can be achieved in a tightly-coupled environment. Moreover, the underlying speed of current large processor models more than compensates for the overhead of locking facilities.
- As more I-stream engines are included in a multiprocessing environment, the chance of contention (waiting for a locked resource) increases. However, this is more than offset by the performance improvement that is gained by adding these I-stream engines.

TPF System Loosely Coupled Multiprocessing

Loosely coupled multiprocessing within the TPF environment refers to a set of central processing complexes (CPCs) each with its own private main storage, and all of which share a set of module CUs to access a shared database. This form of multiprocessing is represented, primarily, by the database support and the underlying I/O services offered by the TPF system.

When several CPCs attempt to simultaneously modify the same module record, an XLF restricts record updating to only one CPC at a time.

For such a mechanism to be effective, the performance overhead to determine if more than one CPC wants to update the record must be minimized and the probability of multiple CPCs wanting to simultaneously update the same record should be low. Both of these conditions are met in the TPF environment.

Observe, as with tightly coupled multiprocessing, loosely coupled multiprocessing comes at some performance expense. Because modules are shared, module CUs or coupling facilities (CFs) must pay the performance expense of setting and checking lock indicators. A CPC must occasionally wait for a record that is currently locked. However, multiprogramming is employed during these waiting periods. Message processing rates in a loosely coupled environment of *n* CPCs are never equal to *n* times the message processing rate of a single CPC using module CUs that have neither the locking facility nor the corresponding system software that controls the locks.

TPF System Coupling Facility Support

Coupling facility (CF) support provides data sharing capabilities that allow TPF routines, subsystems, system products, and applications running in a processor configuration to use a CF for high-availability data sharing. A *CF* is an IBM processor (sometimes referred to as a central processing complex (CPC)) that is used to centralize storage for all attached processors in a processor configuration by providing shared storage and shared storage management functions. CF support provides connectivity to a CF for use by TPF system functions.

You can add one or more CFs from a TPF processor to the processor configuration. Applications can connect to CF list or cache structures on CFs that have been added to the processor configuration. A *CF list structure* is a named piece of storage on a CF that enables users to share information organized as entries on a set of lists or queues. A *CF cache structure* is a named piece of storage on the CF that enables users to share information and allows high-performance sharing of frequently referenced data. A *user* refers to an application or an instance of an application using connection services to access a CF structure.The first connect request issued to a particular CF structure causes that structure to be allocated before establishing the connection.

An application that connects to a CF list structure can monitor individual lists to determine when list entries have been created on that list. When a list changes from empty state to nonempty state (that is, when a list entry is added to a previously empty list), an application-defined exit is called. This eliminates the need for application polling of lists and simplifies programming requirements.

An application that connects to a CF cache structure can automatically notify affected users when shared data in the cache is changed. The application can also determine whether the local copy of shared data is valid by checking system-maintained validity indicators. See *TPF Database Reference* for more information about CF cache structures.

When an application no longer requires access to a CF structure, the application can disconnect from the CF structure. Depending on the parameters specified when the CF structure was allocated, a disconnect by the last connector to a CF structure either causes deallocation of the CF structure or allows it to remain allocated for subsequent connections to occur.

A CF may be added to multiple processor configurations in a TPF multiprocessor complex. A CF that has been added to a TPF processor configuration may ***not*** be shared with any other processor that is not in the TPF multiprocessor complex even if that other processor is also running the TPF system.

Coupling Facility Record Lock Support

The limited lock facility (LLF) and the concurrency filter lock facility (CFLF), which are two external lock facilities (XLFs) supported by the TPF system, were required to control access to data shared by two or more processors in a loosely coupled complex. CF record lock support provides the option of using one or more CFs as XLFs.

CF record lock support offers significant flexibility for using CFs as XLFs in your locking configuration. The CFs in your locking configuration can be used either in addition to or instead of LLF and CFLFs. In addition, the CFs in your locking configuration can be used simultaneously for nonlocking workloads. The use of CFs

in a locking configuration can eliminate the need for LLF or CFLF, giving you greater flexibility when selecting and implementing new module control units (CUs).

You can use CF record lock support to control access to data that is currently shared by two or more processors in a loosely coupled complex by now directing that workload to a CF. CF record lock support uses a logical locking mechanism as a means for serializing data access and processing steps to ensure the consistency and integrity of data records.

Logical Record Cache Support

A *logical record cache* provides you with high-speed access to data that enables you to develop data sharing programs with improved performance. A logical record cache can be processor shared or processor unique. Processor shared logical record caches exploit CF cache support to operate in a loosely coupled complex. With logical record cache support you can use a processor shared logical record cache for the following:

- For data consistency, which ensures the validity of the data that is shared
- To keep track of data that resides in permanent and local storage but is not stored in the CF cache structure itself.

See *TPF Application Programming* for more information about logical record caches.

In addition, you can access and manage logical record caches by using the ZCACH command, and you can manage CF cache structures by using the ZCFCH command. See *TPF Operations* for more information about the ZCACH and ZCFCH commands.

Multiprocessing and Multiprogramming Observations (Summary)

A few observations on tightly coupled multiprocessing, loosely coupled multiprocessing, and multiprogramming within the TPF architecture, are instructive:

- In a loosely coupled complex in which all CPCs are uniprocessors, there is no need for the tightly coupled multiprocessing system programming constructs. In this case, each CPC is a single I-stream engine with a private main storage where separate copies of programs are processed.
- In an environment of a single CPC that consists of an ESA configuration of multiple I-stream engines, there is no need for an XLF, because all module data is directed to a single CPC where it is managed in the TPF system by a single I-stream engine.
- Loosely coupled is more accurately described as the synchronization of multiple CPCs sharing a set of modules, where each CPC can be composed of multiple I-stream engines. From the viewpoint of module synchronization, each CPC is a single I-stream engine.
- A communication mechanism, called interprocessor communications (IPC), coordinates activity among CPCs in a single loosely coupled complex by using the Multi-Processor Interconnect Facility (MPIF) a feature.
- A TPF system can be configured without loosely coupled multiprocessing. However, tightly coupled multiprocessing constructs are always in place.
- The basic component of performance in the TPF system, multiprogramming, is basic to the design of the TPF system. This makes it possible for several Entries to be in progress at any one time, thereby enhancing performance.

TPF System Structural Characteristics

One of the problems of documenting anything is related to the use of descriptive information to discuss real objects, such as the TPF system. The descriptive information is used to discuss the reality, but it is ***not*** reality. Reality, in terms of the TPF system, may be found on the assembled listing of code, by looking at bits on a disk drive, by observing an operator start an initial program load (IPL), by watching an operator mount a tape, by watching end users at remote terminals or workstations, or by following the electrons in an I-stream engine. Although these things ***represent*** the real thing, it is a tedious way to learn about the system. Therefore, descriptive information is used that allows a more concise ***abstraction*** of the real thing. A control diagram is useful for introducing the control structure of the TPF system within the framework of following the flow of a message through the system.

TPF System Control Diagrams

Two types of control diagrams are used to show the structural characteristics of the TPF system:

- Control structure diagram
- Control transfer diagram.

The main distinction between these types of diagrams is that a control structure diagram does not have arrows, while a control transfer diagram does.

Figure 14 on page 54 is a control structure diagram. A node in a control diagram is interpreted in the following sense:

- A system element, represented by a node, is a sequential process; that is, a sequence of operations carried out one at a time. One I-stream engine interprets one process at a time.
- Within any I-stream engine, only one node is in control at any *instant of time*. Control means being processed by an I-stream engine, and instant of time means the interval of time necessary to complete an operation (ESA instruction).
- Nodes can represent concurrent processes. Each process (node) must keep other processes (nodes) in the structure informed of the use of shared resources. Some of this awareness is automatically handled by the I-stream engine, for example, through the use of program status words (PSW); however, much of this awareness is saved in system control blocks.

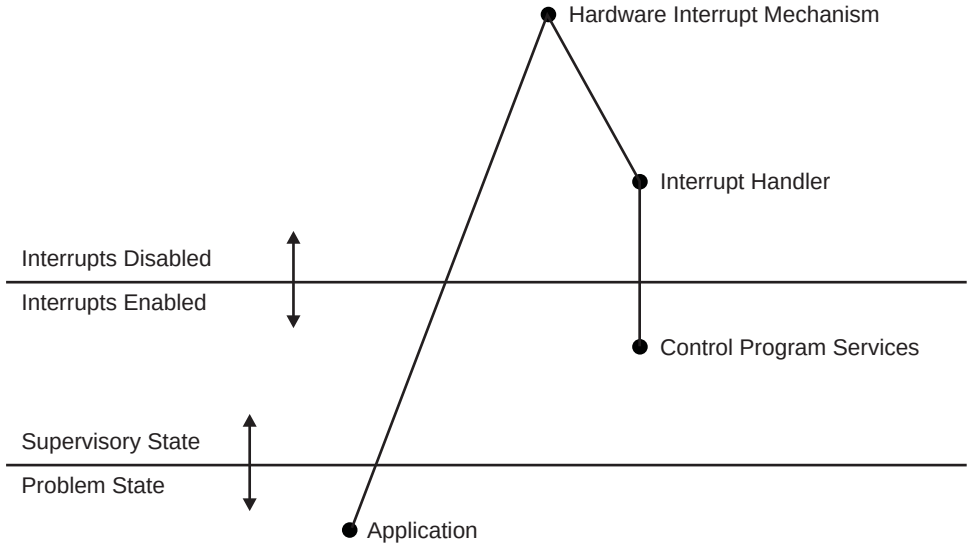


Figure 14. Simple System Control Structure Diagram

The control program in the TPF system manages the time dependencies caused by sharing resources among concurrent and simultaneous processes. Concurrent processes can be explained as when the first operation of one process is started before the last operation of another process is completed. Simultaneous processes are defined as when the operation of several processes can occur during the same moments of time. Notice that simultaneous processes are concurrent processes, but the opposite is not necessarily true.

A control structure diagram emphasizes the structure within a single I-stream engine. Within a central processing complex (CPC), such a control structure is associated with each I-stream engine. Each of the processes (nodes) has the potential for invoking a TPF processor lock on a resource which, if accessed by another I-stream engine, generally delays further progress on the other I-stream engine. However, simply invoking a TPF processor lock does not mean that all other I-stream engines are delayed. Simultaneous access to the same resource is required to cause an I-stream engine delay. Delays because of processor locking are minimal.

The lines connecting the nodes (processes) do not necessarily represent simple transfer instructions. They are used to show that the connected elements must occasionally invoke locking mechanisms in order to do their processing; otherwise, the TPF system becomes confused. Invoking a processor lock always creates the potential for an I-stream engine control diagram to become connected to any other I-stream engine control diagram; for example, when two I-stream engines attempt to lock at exactly the same instance of time. Within the context of control diagrams, the interrupt mask bits in a PSW also represent a lock; this locking is done to synchronize processing between an I-stream engine and the engines within a channel subsystem.

Consider the following hypothetical situation that occurs in the simple system shown in Figure 14. When an application requests a system service that results in a supervisor call (SVC) instruction to perform disk I/O (input/output), the application expects control back when the I/O is started, placed on a queue of work to be performed, or completed.

During the interval that the SVC instruction is being interpreted by the I-stream engine, an interrupt-causing signal occurs as a result of random input from the communications facilities. Also, a timer causes an external interrupt to occur. The hardware interrupt mechanism stacks the interrupts (as described under “Interrupt Processing” on page 33). The SVC service routine allows each of these interrupts to be processed in turn and ultimately builds the channel commands for initiating the I/O. Some of the service processing can be done with more interrupts permitted. However, there are intervals of time, such as when the common I/O routines are invoked, when further I/O must be inhibited; otherwise, status information could be lost. This is done with a set system mask (SSM) instruction.

Finally, the control program service routine loads the old SVC PSW, which passes control ***directly*** back to the application program that requested the system service.

Now look at this activity and its relationship to the control structure diagram. Although the control program service routine transfers directly to the application program, this is only possible because:

- The hardware interrupt mechanism stacks simultaneous interrupts and status information.
- The TPF system uses queues.
- The TPF system uses the set system mask (SSM) instruction.

All these things happen at different points in the processing but enable the TPF system to finally ***return control*** to the application from the service routine. The indicators in the hardware are maintained in a manner such that:

- Interrupts are blocked when appropriate
- Certain critical code is executed completely without interruption.

So even though the transfers can be as given in Figure 15 on page 56, the control structure should be viewed as given in Figure 14 on page 54. For this reason the lines in the control structure diagram do ***not*** have arrows, which is meant to emphasize that they are ***not*** necessarily transfer instructions but represent queues or status.

The nodes in a control diagram represent sequential processes.

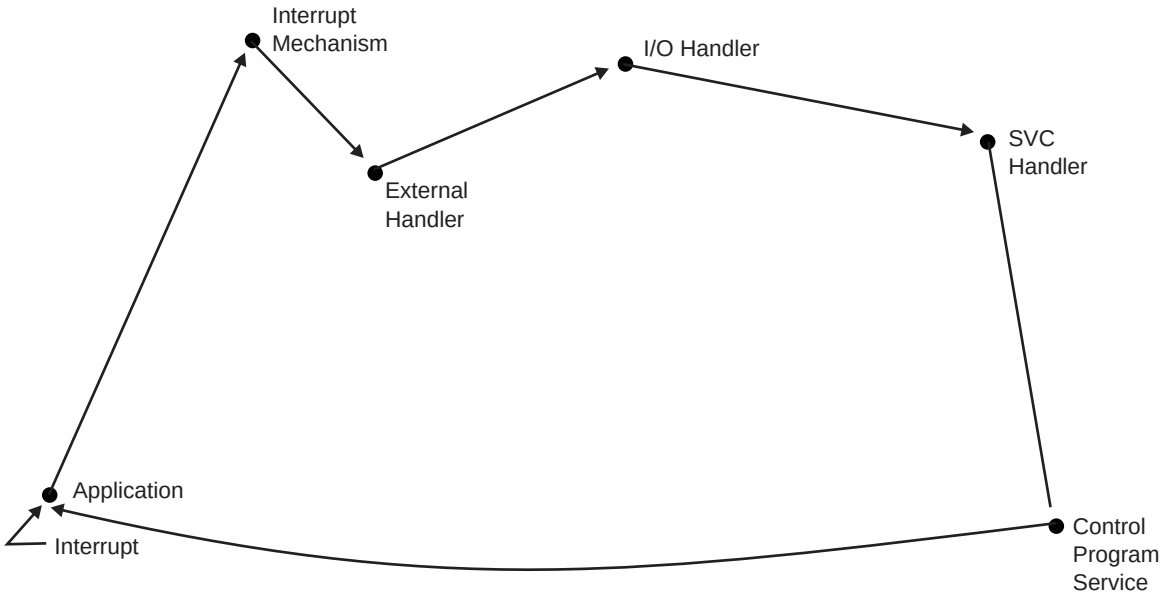


Figure 15. A Control Transfer Path

The TPF System Programming Terminology

Up to this point, the programming elements in the TPF system have usually been called the TPF system. However, because we are about to discuss the details of the TPF system, it is necessary to further define the programming elements of the TPF system.

So far, most of the references to the TPF system actually refer to what is called the control program. The control program is the collection of programs that handles interrupts, coordinates concurrent processes, requests input from communications networks, provides system services, and so on.

"TPF Entry" on page 39 introduced the concept of an Entry, which is associated with an input message from an end user. In order for Entries to be processed concurrently, an area of main storage is assigned to each Entry for use as an application work area and system work area. This area is called an entry control block, or commonly called an ECB.

Control Structure for the TPF System Defined

The TPF system has a very simple control structure that is shown in Figure 16 on page 57. The nodes of the control structure diagram represent:

- The hardware interrupt mechanism, as described under "Interrupt Processing" on page 33.
- The first level interrupt handler, which represents the routines that receive control upon a non-SVC interrupt (external, machine check, program, restart, or I/O).
- The CPU loop that dispatches programs for execution on an I-stream engine.
- The macro decoder, which begins the interpretation of the parameters received as the result of an SVC interrupt.

This is how an application makes requests for system services. The macro decoder essentially switches to system services routines that generally: (1) initiate I/O and (2) queue requests as a result of timing dependencies.

- Communications control, which receives messages from the communications facilities.
- OPZERO, which creates an ECB when the I/O operations for reading a new input message are completed. This act creates the TPF application process called an Entry.
- COMM SOURCE, which locates the application program segment necessary to begin the interpretation of the message content
- The application programs, which interpret the semantic content of a message, generally resulting in accesses to the database and in generating a response message.

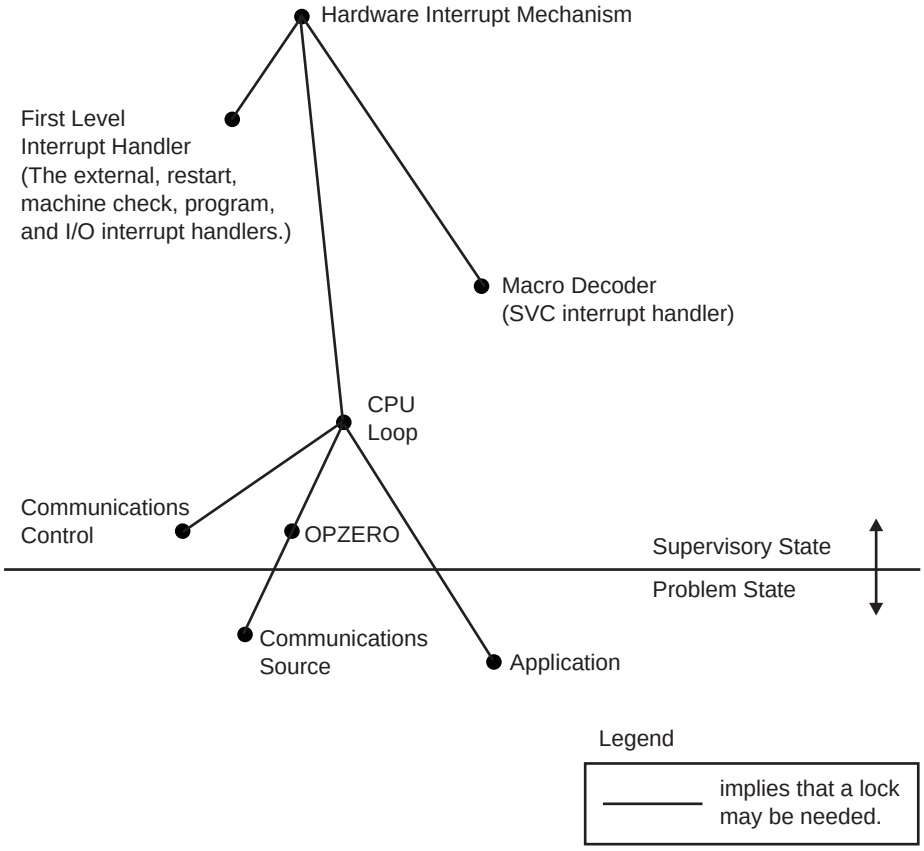


Figure 16. TPF Control Structure Diagram

Message Processing Overview

Tracing the flow of a message is a pragmatic way to introduce conventions and terminology used by the TPF system and to make the abstract control concepts a reality. Figure 17 on page 60 represents some additional detail to the contents of the control structure diagram in Figure 16. The boxes and solid lines in Figure 17, as a matter of fact, represent the control structure diagram in Figure 16. Additional conventions used in Figure 17 follow:

- Solid lines represent a line of control as described earlier in this chapter. Notice the solid lines do not have arrows, which is meant to emphasize that control lines are not necessarily branching paths.
- Dashed lines are used to show certain transfer paths (branches). This is meant to show that these transfers (branches) are made without losing control information as a result of some previous sequential (because interrupts are inhibited) processing on a queue or status indicator.
- The double lines represent data paths that become involved in maintaining control.

The principal idea of the TPF system design is to place the TPF system into a state where the system is requesting message traffic (in the form of input messages) from the communications facilities, and updates to the database and outgoing messages (as responses to agents) are a result of application processing.

The CPU loop inspects the cross, ready, input, and deferred work lists and checks status indicators. Among other things, some of the checking of status indicators causes input messages to arrive in the TPF system.

An input message causes an Entry to be created and application program segments to be dispatched for processing the input message. The Entry makes requests of the control program for system services such as:
- Input and output (I/O)
- Storage allocation (both file and main storage)
- Program fetches (called Enter/Back in the TPF system)
- Release of control.

Many of the system services requested by an Entry have other system services implied. For example, if application program segment X calls application program segment Y, and Y is not already in main storage, then the system must:
- Obtain a working storage block in main storage for Y
- Issue the I/O commands necessary to retrieve Y from file storage
- Try to find another Entry that can be started or continued during the delay caused by the I/O.

Execution Summary

The TPF system processing shown in Figure 17 on page 60 consists of the following steps:

1. Starting with the assumption that the system is initialized and the CPU loop is in control (processing), the CPU loop, as part of its processing, requests input messages from communication controllers in the communication facilities.
2. Input messages received from the communication facilities are ultimately placed in main storage buffers. As a result, the I/O interrupt handler invokes system programs and puts work items (associated with the input messages) on the processing queues (lists).
3. When the CPU loop finds an item on the input list, control is given to OPZERO. OPZERO creates an ECB and gives control to COMM SOURCE, which selects the application program segment necessary to begin interpreting the input message.
4. As SVC interrupts are received, the requests for system services are processed. Often this involves I/O.
5. When the request for I/O is serviced:
   a. Put the I/O request on an I/O device queue.

b. If the control program finds that the application currently in control must wait for the activity associated with the requested system service to complete, the Entry is suspended and the CPU loop receives control again to interrogate the lists and check for the arrival of more input messages.

6. As I/O completes, work items are placed on the ready list.
7. The CPU loop can:
   a. Return to a previously suspended Entry, which is an item on the ready list
   b. Start a new application (Entry) as the result of an item on the input list
   c. Continue looping until input messages arrive.

In terms of sheer logic, the purpose of the control program is to reach the CPU loop with nothing to do (that is, for the CPU loop to process empty work lists). If this can be accomplished with the end users receiving their responses in a timely fashion, then, relative to the end user, the system is performing as expected. The trade-offs in a performance-oriented system are very sensitive. On one hand, if the system is actually in the CPU loop with nothing to do, then there is excess computing power. However, if the computing power is over utilized, response time increases. The TPF system facility called data collection and reduction is used to *tune* the system and cope with the sensitive balance between utilization of resources and response to the users.

In any event, the execution of an application program segment (an Entry) results in requests for system services (through SVC interrupts) that can cause a delay. During this delay time, the CPU loop can give control of the I-stream engine to a different Entry.

Random (non-SVC) interrupts can be received that have nothing to do with the Entry currently in control. In the TPF system, the random interruption of an Entry does not cause the control program to switch to the processing of a different Entry. Instead, the control program places the information about any interrupts in a queue and returns control to the Entry that was interrupted. This is a significant attribute of the TPF system.

So, a basic premise of TPF system design is that no interval of processing by an Entry is assumed to ever require the instruction execution capability of an I-stream engine for more than a relatively small amount of time. This is a fundamental design decision that eliminates the need for complex algorithms for *time-slicing* code that is found in many *time-sharing* systems. This makes it difficult (but not impossible) to write applications to run under the TPF system that are compute bound.

Several processing intervals are usually required to accomplish the processing required by an Entry. A processing interval is, for example, measured from the time an Entry receives control until the Entry gives up control to wait for an I/O completion or because it has finished processing. I/O wait time does not count as processing time.

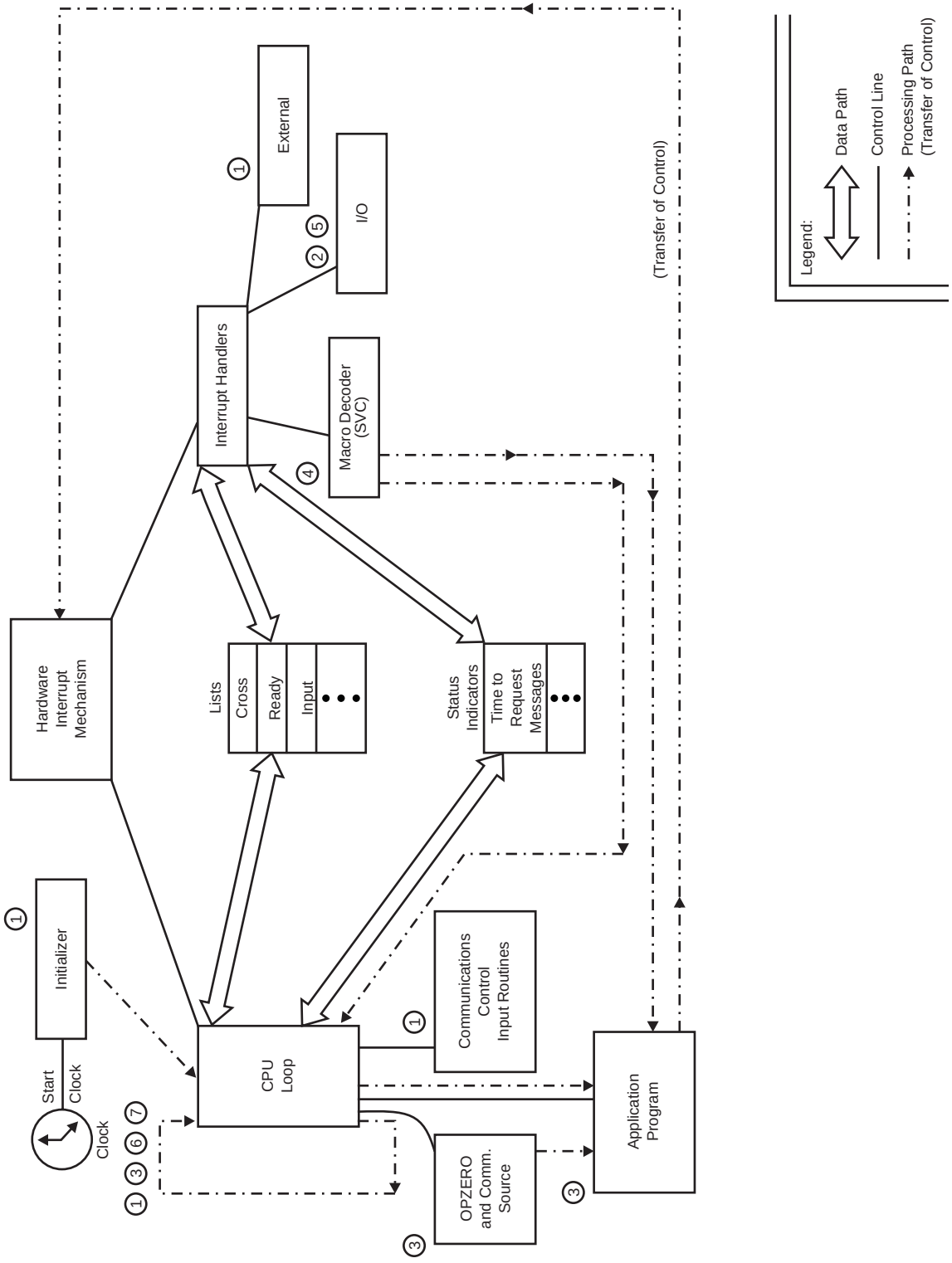


Figure 17. Normal TPF System Execution Overview

System Initialization

The entire process of initializing the TPF system is indicated by the box in Figure 17 on page 60 labeled *Initializer*. Initializing the TPF system includes:

- Formatting the system direct access storage devices (DASDs)
- Loading the system programs to the system DASD
- Initializing the system sensitive data (contained in data structures called keypoints)
- Invoking the initial program load (IPL) procedure that loads the control program into main storage for the purpose of starting (or restarting) system activity.

CPU Loop (Dispatching Work)

The TPF system uses a system program called the CPU loop to select an Entry for execution by an I-stream engine. Sometimes the term system task dispatcher is used rather than CPU loop. In a central processing complex (CPC) with multiple I-stream engines, the CPU loop (operating within an I-stream engine) continuously inspects its cross list to which another I-stream engine may have added work.

Although the phrase *system task dispatcher* is sometimes used in TPF documentation, the term *CPU loop* is pervasive. Task is a term that has the connotation of an application process found in the IBM MVS operating system. A TPF Entry is structured differently from an MVS task. Rather than being called the *system task dispatcher*, a better name would be *system entry dispatcher* or simply *dispatcher*. In the face of this dilemma, the term *CPU loop* is used in this publication.

The order of processing priority is determined by the sequence in which the CPU loop interrogates queues that identify work items to be dispatched. The term *list* refers to the CPU loop queues. Unfortunately, this term is also used to refer to tables that are not queues. As a result of the long history of the TPF system, the vernacular phrase *CPU loop list* is also used in this publication to refer to a queue used to dispatch work to an I-stream engine.

The CPU loop lists, in order of processing priority, are:

1. Cross list — Used for dispatching entries between I-stream engines. (This is the highest priority queue.)
2. Ready list — Used to return control to an Entry already created as the result of some completed system activity.
3. Input list — Used to dispatch a new Entry for processing an input message.
4. Deferred list — Used to delay the execution of an Entry. (This is the lowest priority queue.)

Through linkage conventions, these queues point to all the input necessary for an application to initially process or continue processing an input message.

In addition to the four main CPU loop lists there are two secondary lists:

- Virtual file access count (VCT) list: Contains entries that were forced to give up control after exceeding a system resource threshold, one of which is the number of virtual file access (VFA) record accesses.
- Suspend list: Contains entries that were suspended after exceeding a system resource threshold. The suspend list is used for the LODIC macro when the availability of a particular block type has fallen below a defined shutdown level, or the TMSLC macro when the entry has run for a defined time limit.

The VCT and suspend lists are checked once every pass through all items on the input list.

In principle, the CPU loop is a set of programs with pointers to unique processing work lists (CPU loop lists) depending on which I-stream engine the CPU loop is running. All the lists, except the cross list, are private to an I-stream engine. The cross list is used to move work between I-stream engines. The CPU loop lists are located by pointers anchored in Page 0 (for each I-stream engine).

An Entry and an item on a CPU loop list, although closely related, are not the same thing. An item on a CPU loop list points to a system service routine that must be invoked before starting or returning to an Entry. This distinction is necessary to describe some important details required to understand the TPF structure. When you get to the detailed system documentation, an *entry* on a list may not be distinguished from a TPF application process called an *Entry*.

Control program components associated with message processing place items on the CPU loop lists:

- Communications control input routines place items (corresponding to input messages) on the main I-stream engine input list. By design, these routines run only on the main I-stream engine.

  In a uniprocessor environment, Multi-Processor Interconnect Facility (MPIF) input routines also place items on the main I-stream engine input list. In a multiprocessor environment, however, the MPIF input routines run only on I-stream engine 2 (known as the *MPIF I-stream engine*), and MPIF input items are placed on the input list for this I-stream engine. MPIF I/O is considered to be high priority and, therefore, is handled by an I-stream engine that is different from the one that handles all the other non-DASD-related I/O for the TPF system.
- System service routines, invoked as a result of interrupts caused by I/O completion, place items on the ready list.
- COMM SOURCE, running in the main I-stream engine, dispatches an Entry to an application I-stream engine by placing the Entry on the cross list of the (destination) application I-stream engine. When the application I-stream engine finds the Entry on its cross list, the Entry is moved to its ready list.

Whenever the ready or input lists are not empty, the CPU loop merely selects the work identified by the first item on one of these lists, giving priority to the ready list. (The deferred, VCT, and suspend lists are not needed to trace a normal message through the system and are not emphasized in this overview.)

Operation Zero Program (OPZERO)

OPZERO refers to a collection of system programs associated with communications control in the TPF system. Essentially, there is one OPZERO program per type of communication facility (where a type is loosely equivalent to a communications protocol).

OPZERO creates an entry control block (ECB) and associates the input message with the ECB. OPZERO then passes control (and the ECB) to COMM SOURCE (Communications Source Program) to continue input message processing.

In the TPF system, OPZERO is functionally considered to be part of communications control. There is more detail about OPZERO in Data Communications.

Communications Source Program (COMM SOURCE)

COMM SOURCE refers to a collection of system programs that transform input messages from their individual protocol-dependent formats into a common system

format that is recognizable by applications. This relieves the application from any awareness of the various protocols in the TPF system.

COMM SOURCE uses system tables to determine the application program segment to pass control to for the continuation of input message processing.

In the TPF system, COMM SOURCE is considered to be functionally part of communications control. There is more detail about COMM SOURCE in Data Communications.

Message Flow Through the TPF System

This processing necessary to respond to an input message is described in more detail in this section. This *message trace* is meant to be a pragmatic introduction to TPF system conventions and structure. Keep in mind that this section describes the flow of a single message through the system in contrast to a TPF transaction (see "Transaction Defined" on page 11).

The arrangement of this section is important. Figure 18 on page 64 contains a conventional flow diagram (that is, the lines are meant to represent processing sequence). The ***circled numbers*** identify the sequence of processing and correspond to the numbers given in the following text. An ***oval*** symbol is used to show a point of processing that is taken when an I-stream engine is interrupted; the point of processing is dependent on the type of interruption. Occasionally, two different numbers appear side by side, which means the process has returned to the system component where the different numbers are given. Also, the same number appears at different points on the chart. This means that various components are involved with the same processing step.

Important detail is included in the following sections that trace a single message, including discussion of tightly coupled multiprocessing where necessary. This detail keeps the essence of message processing relatively concise. Upon first reading, it should be possible to obtain an accurate conceptual idea of message processing without bothering with too much detail. This trace of a message through the system and the remaining sections of this chapter introduce the management of resources provided by the TPF system.

Now, refer to Figure 18 on page 64 as you read the following steps that trace the flow of a message.

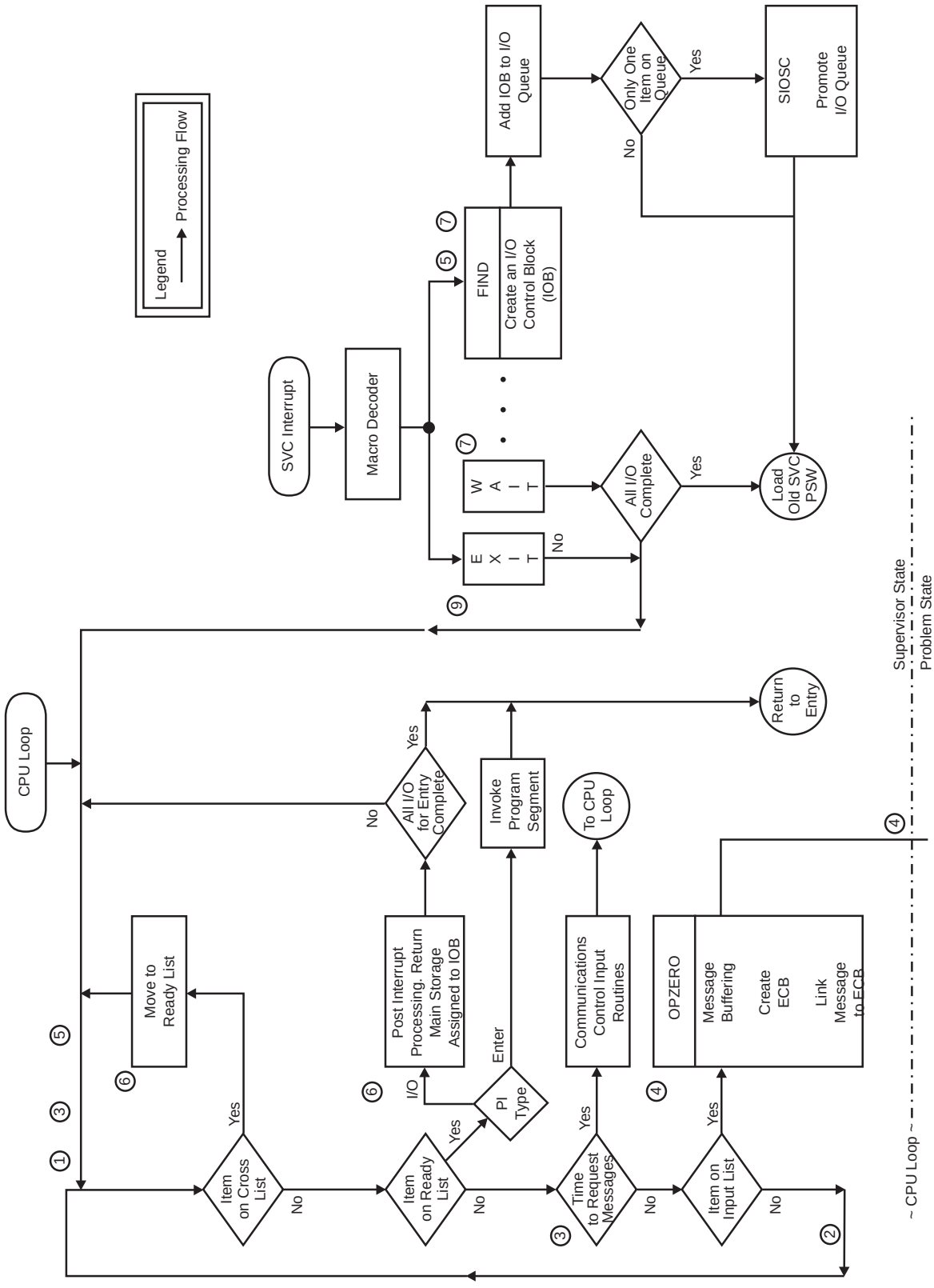


Figure 18. Message Processing Flow Diagram (Part 1 of 3)

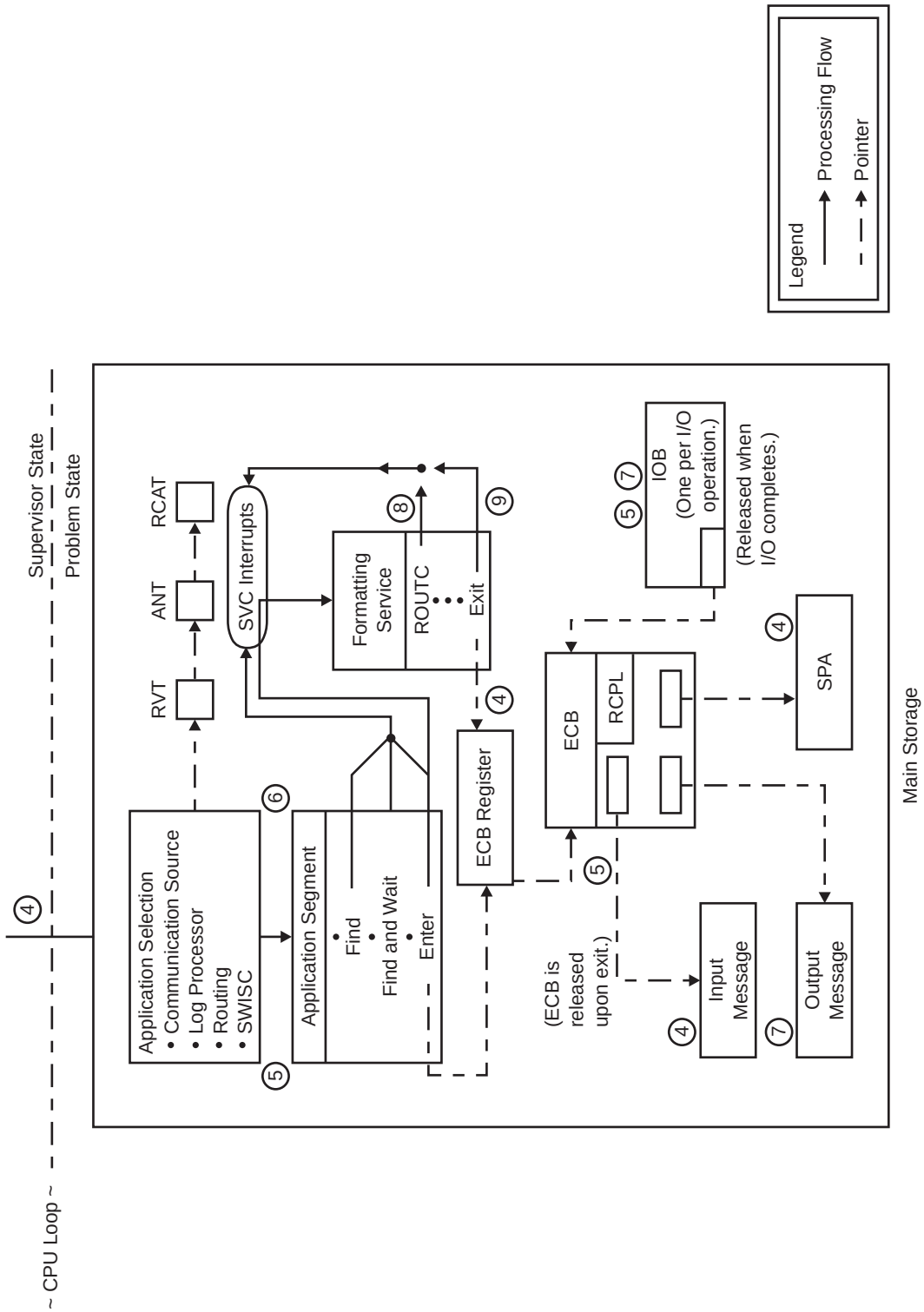


Figure 18. Message Processing Flow Diagram (Part 2 of 3)

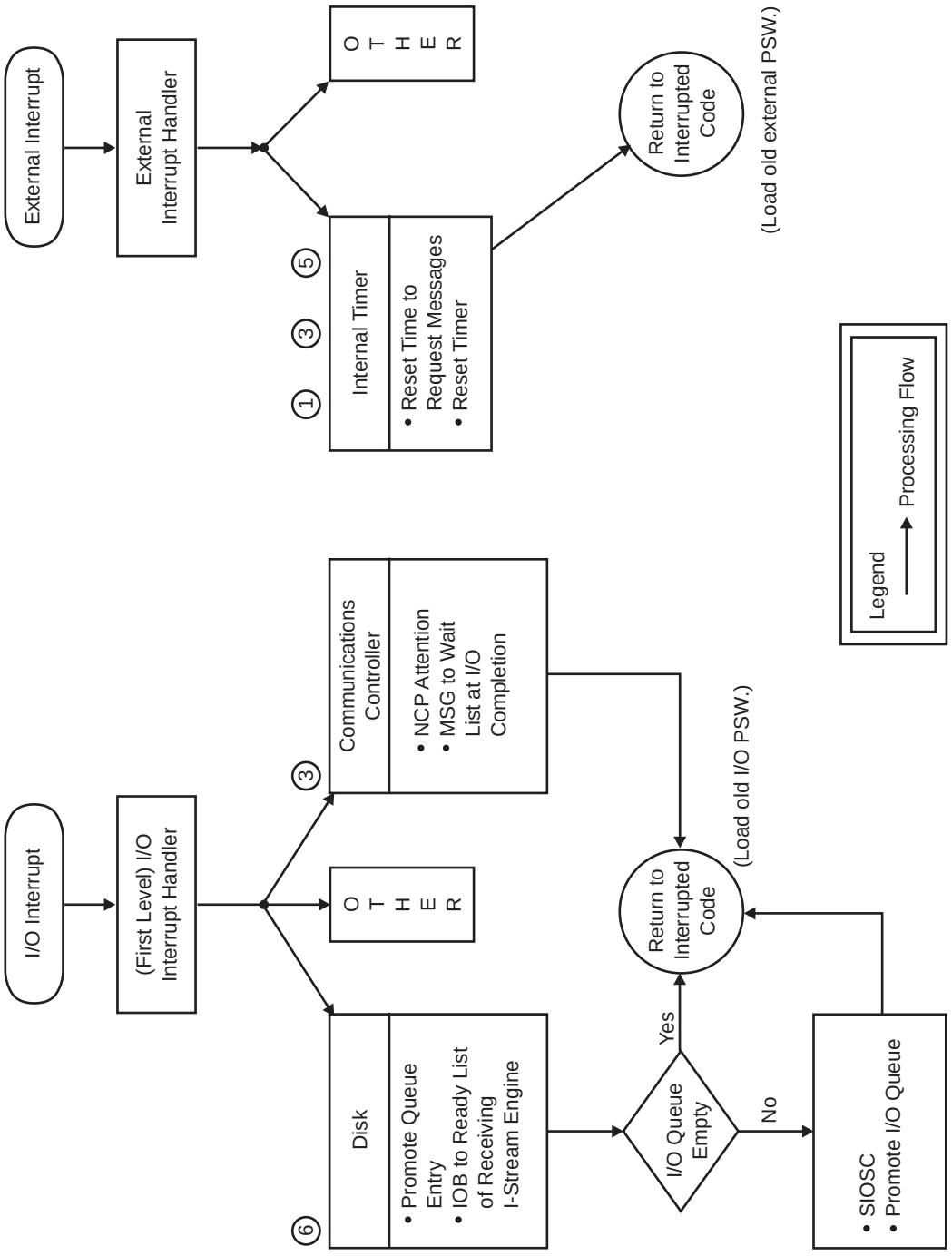


Figure 18. Message Processing Flow Diagram (Part 3 of 3)

Step 1. The System is Initialized

Start by assuming that the TPF system has been initialized and the CPU loop is in control.

Step 2. CPU Loop Checks for Work on the Cross, Ready, and Input Lists

The CPU loop finds nothing on the cross list, ready list, and input list, and because the timer has not yet caused an external interrupt, continues looping.

Step 3. Input Messages Arrive

In the TPF system, there are several ways that input messages arrive:

- By actions initiated by the CPU loop
- By actions initiated by the I/O interrupt handler

Data Communications discusses this topic in more detail. However, for now, the actions initiated by the CPU loop are used for this discussion.

The CPU loop finds that it is time to request input messages and invokes the communications control input routine, which starts an I/O operation to read the data from the communication network.

Observe that the willingness to accept an unlimited number of input messages could exhaust main storage. The system avoids this problem by placing an upper bound on input messages, known as a shutdown level. In a properly configured TPF system, the input list shutdown level only occurs if the system is driven beyond its design limits, and normally shows up as response time degradation. A goal of most TPF installations is to tune the system so that the shutdown level is virtually never observed.

Step 4. Create an ECB and Select an Application

The CPU loop finds an item on the input list. The following functions are accomplished through the programs OPZERO and COMM SOURCE.

Step 4a. Message Preprocessing

The format of an input message arriving from a communications facility is specific to a communication protocol. (There are multiple protocols supported both on the SNA and non-SNA networks.) For each communication protocol, there is an associated OPZERO program that handles the details peculiar to that communication protocol. However, for the present, we can ignore these details. From this point on, the term OPZERO refers to the common functions of the various OPZERO programs. Furthermore, assume that an input message fits into a single storage block so that our discussion does not need to include such details as chaining a message and the isolation of separate messages from a collection of messages that can be in an input buffer.

The details of communication protocols and message preprocessing performed in OPZERO and COMM SOURCE are discussed in Data Communications.

Step 4b. OPZERO Creates and Initializes an ECB

The ECB is a main storage area, private to a specific input message or, more accurately, to a specific Entry. A subroutine invoked by OPZERO creates the ECB that defines the Entry. A pointer to the input message is placed in the ECB.

A transition is made from the supervisor state of the control program to the problem state of the application (Entry) environment, where all the system services designed to work in conjunction with an ECB can be invoked. At this point, the ECB register is loaded (initialized) with the address of the ECB. The ECB register is an Entry's indirect reference to a private main storage block — the cornerstone of the

application program reentrant structure. The reentrant programs that refer to private data through the use of an ECB are called ECB-controlled programs (or E-type programs).

When processing an input message, COMM SOURCE is the first system program that is ECB-controlled.

Step 4c. COMM SOURCE Invokes the Application

COMM SOURCE places the input message into the common message format used by the application environment. This format is known as AMSG.

COMM SOURCE initializes the routing control parameter list (RCPL) associated with an input message throughout its processing.

The ECB associated with the input message is placed on the cross list of an appropriate I-stream engine by COMM SOURCE through an SWISC macro request.

The details of COMM SOURCE is a topic in Data Communications.

Step 4d. An Application is Selected

The programs invoked by COMM SOURCE are frequently installation-written *transaction processing editors* that select one of the many message processing programs. If the processing of the current message is dependent upon previous message processing, the transaction processing editor can retrieve the terminal control block associated with the source of the input message.

The combined functions performed by COMM SOURCE, the log processor and the retrieval of the terminal control block are in the box labeled *Application Selection* in Figures 12 and 18.

In summary, messages move in (and out of) a central processing complex (CPC) through the use of interrupt handling routines for communication controllers, the communications control input routines, OPZERO, COMM SOURCE, and the macro decoder (for output). Although these functions occur at different times, all except the macro decoder are considered to be part of the communications control function.

Step 5. Fetch Application Program from File

Generally, some segment of the application program package necessary for processing the message must be retrieved from file. If this is the case, the appropriate I/O is started for accessing the necessary program segment. (See “Enter/Back (Program Linkage)” on page 78 for information about the procedure for invoking file resident application programs.)

At this point, further processing related to this input message is delayed until the I/O completes. During this delay, the CPU loop can process other work (Entries).

Step 6. Starting Program

Eventually, the program segment needed to continue message processing is read into main storage. As a result of this completed I/O, an item is ultimately placed on the ready list of the I-stream engine that initiated the request. Moving I/O work between two I-stream engines is described in “Action on the Cross List (Switching I-Stream Engines)” on page 84. When the CPU loop finds this item on the ready list, the suspended application Entry once again receives control. The ECB, terminal

control block, and input message are still in main storage at this point. For each I/O delay, the CPU loop continues its processing as described in “CPU Loop (Dispatching Work)” on page 61.

Step 7. Running Applications

Assuming the input message requires a response, an Entry obtains a block of main storage in which the output message is built. During the processing of an input message, the Entry makes (I/O) requests to access file storage by using find and file macro requests. If the Entry informs the system that further processing should be delayed until the I/O is complete, through the WAITC macro request, this type of delay is handled like the delay described in Steps 5 and 6. (This is the delay shown as an “I/O gap” in Figure 5 on page 15.)

Step 8. Sending the Reply

When processing of the input message is complete, the output message is transmitted to the end user over communication facilities. The Entry can issue a ROUTC macro request for this transmission. This implies I/O processing that is not shown in Figure 18.

An application can request system services to format the output message to accommodate a communication protocol (because of unique device characteristics). The formatting services program is an application interface (API) to the function of message transmission in the TPF system.

Step 9. Release Resources and Cleanup

When all the processing required by an Entry is complete, an EXITC macro is issued. This causes the TPF system to release all main storage utilized for processing the input message, as well as to perform other system housekeeping.

Summary of Message Flow

The flow of a message has hopefully demonstrated that communications control, as well as other I/O support, is an integral part of the TPF system. The intimate relationship between the communications control and the other system control components in the TPF system is a processing philosophy used to improve performance. The introduction of the following components represents the principal interfaces between the communications control and the application environment:

- Message reception and ECB creation by OPZERO
- Application selection by COMM SOURCE and the log processor program
- Formatting and transmission of the output message by a formatting services program (optional).

Some important system functions and detail are not readily identified by the pragmatic approach of describing the flow of a single message. These system facilities and conventions are described in the following sections.

Entry Control Block (ECB) Overview

The entry control block, usually called the ECB, is the cornerstone of the application program reentrant structure in the TPF system. Use of the ECB allows one program to service multiple Entries because an ECB is private to an Entry. Within the ECB, there are work areas that are ordinarily an integral part of a non-reentrant program. Since reentrancy is a basic attribute of a TPF application program, the ECB is the facility provided in support of reentrancy.

The ECB consists of main storage blocks and is the link between the TPF system and the application, as well as between the other program segments (ECB-controlled programs) within the application package. That is, portions of the ECB are defined as an application interface (API).

Format of an ECB

See Figure 19 on page 71 for a diagram of the ECB.

The ECB is 12KB long and is referred to in three 4KB page segments.

- ECB page 1 is primarily used by applications and is the link between the TPF system and the application program (ECB-controlled program). It contains:
  - Fixed application work areas, which are used by the application program

    This is work space for use by application programs in any way it is required. The work areas are unformatted and of fixed sizes. However, if an application requires additional work area beyond what is provided by the ECB, it can be requested from the TPF system.
  - Fixed system work areas
    - Resource interface area, which is used by both the application program and the control program

      This area contains the addresses of file records and working storage blocks that the application program has requested from the TPF system.
    - Entry management area, which is used by the control program

      This area contains status information, a register save area (used by the macro decoder), PSW save area, and other data required for the management of an Entry.
  - User-definable area, which can be used by the application program

    The space is for use by application programs in any way it is required.
- ECB page 2 is primarily for use by the control program

  It is used to hold data related to the Entry that might impact system availability if damaged by the application program, such as control information used in the management of the ECB and working storage blocks.

  There is also a user-definable area in ECB page 2. The difference between this user-definable area and the one in ECB page 1 is that this one offers an additional level of protection because its storage key is not the same as the Entry's. Protection is achieved because the storage protection key must be explicitly changed in order to update this area.
- ECB page 3 is for use only by the TPF system's control program and contains data and tables related to the management and activity of the Entry.

Accessing the ECB

The address of the ECB is in the ECB register, which is part of the interface when control is passed from the control program to the application. OPZERO initializes the ECB register when the ECB is created.

By convention of the TPF system, the ECB register is register 9.

Creation of an ECB

An ECB is created for an Entry by OPZERO as described in "Step 4b. OPZERO Creates and Initializes an ECB" on page 67.

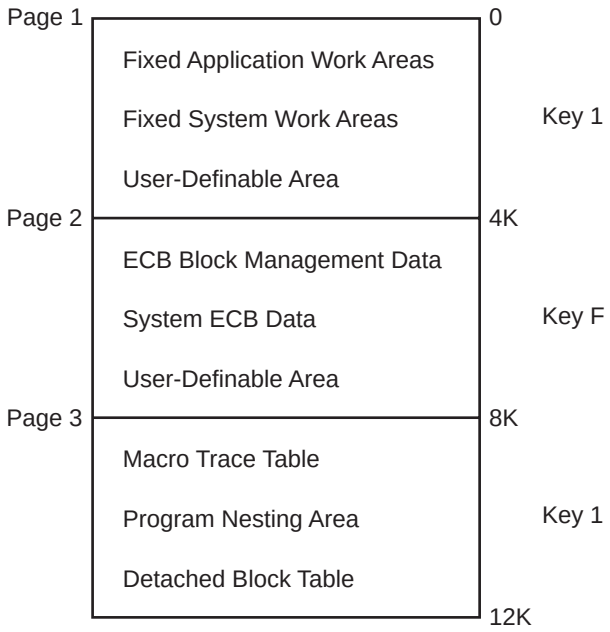


Figure 19. Entry Control Block (ECB)

Data Event Control Block Overview

Before the addition of TPF data event control block (DECB) support, the TPF 4.1 system restricted the number of entry control block (ECB) data levels (D0–DF) that were available for use to 16 (the number of data levels defined in the ECB). A DECB can be used in place of data levels for FIND/FILE-type I/O requests by applications. Although a DECB does not physically reside in an ECB, it contains the same information as standard data levels: a core block reference word (CBRW), a file address reference word (FARW), file address extension words (FAXWs), and a detailed error indicator.

TPF programs can dynamically create and release DECBs by using C++ functions `tpf_decb_create` and `tpf_decb_release`, or the DECBC macro. The number of DECBs that can be created is only limited to the storage that is available in the ECB private area (EPA).

DECB frames are connected to page 2 of the ECB from address field CE2DECBPT. The DECB frames contain multiple DECBs that hold the CBRW and FARW. A DECB may contain the address of a working storage core block. This address (or pointer) is maintained in the CBRW section of the DECB. Even though an address is in the CBRW, this does not mean that the core block is attached. The core block type indicator will not be X'0001' when attached. Figure 20 on page 72 shows the format of a DECB:

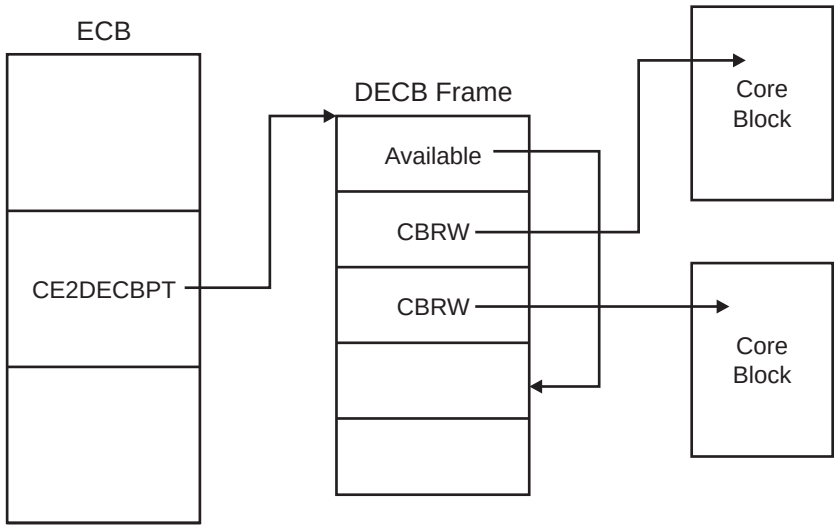


Figure 20. Format of a DECB

For more information about DECBs, see *TPF Application Programming*.

Main Storage Management Overview

Virtual Address Space

The TPF system defines several types of virtual address spaces, which are mappings of main storage:

IPL virtual memory (IVM)	The IVM mapping is all of main storage that can be used (addressed) by a particular I-stream engine except for certain IPL-related data areas that are only mapped on real storage.
System virtual memory (SVM)	The SVM mapping is a variation of the IVM mapping.
ECB virtual memory (EVM)	The EVM mapping is all of main storage that can be used (addressed) by an Entry.

These mappings are shown in Figure 21 on page 73.

An application program (ECB-controlled program) uses the EVM mapping of main storage when processing. The control program uses both the SVM and EVM mappings of main storage depending on the type of service it provides at any given time.

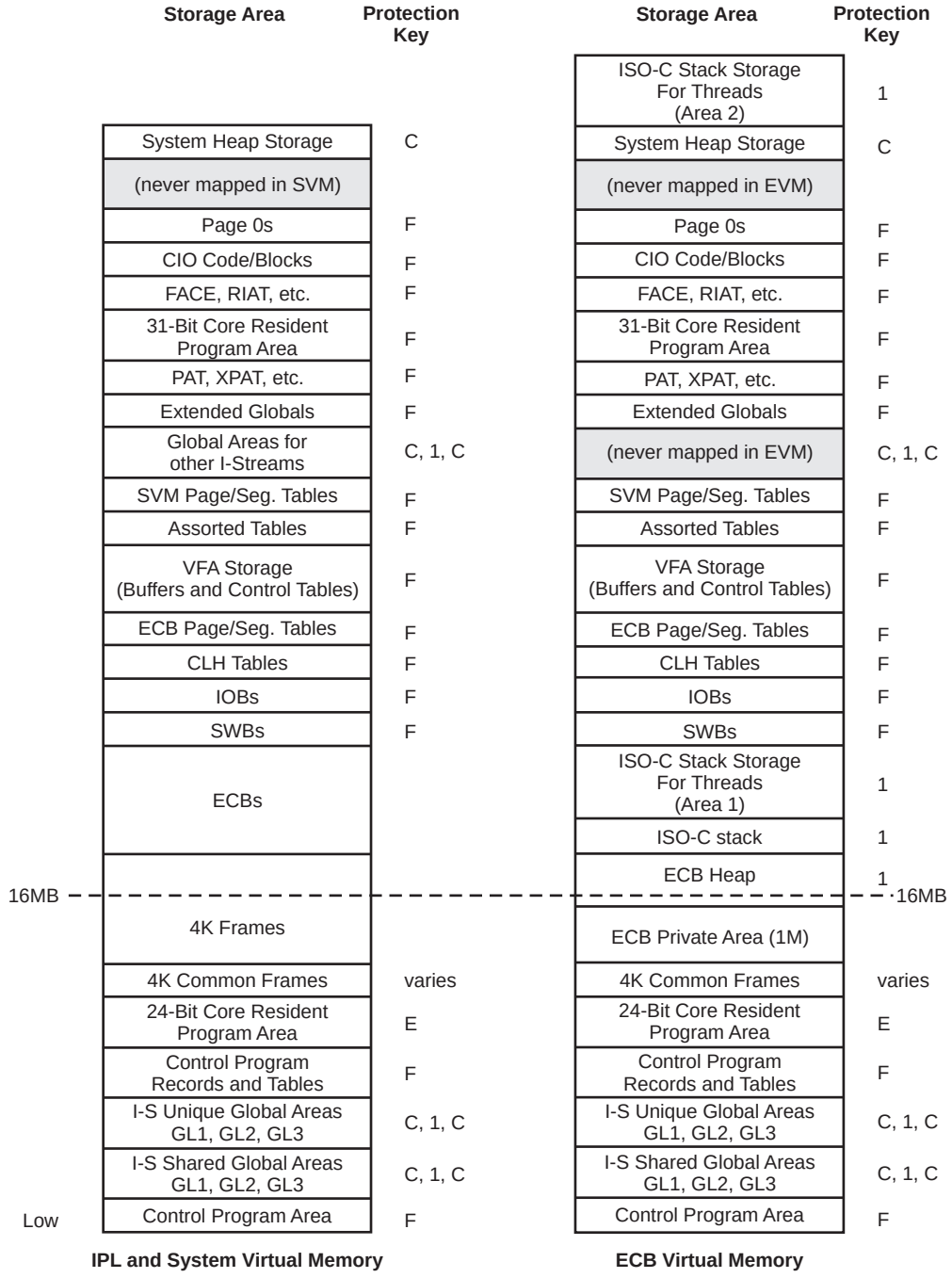


Figure 21. Virtual Storage Layout

Note: TPF's virtual address spaces makes it impossible to use a full 2 gigabytes of real memory.

Fixed Storage and Working Storage

Main storage consists of fixed storage (in earlier versions of the TPF system, fixed storage was known as *permanent core*) and working storage. Fixed storage refers to those areas of main storage whose sizes are determined at system generation (that is, the sizes are not determined dynamically during system operation), such as system data records and tables. The term working storage refers to those areas of

main storage that are (1) available to application programs as system resources and (2) the system control blocks used for managing an Entry.

The maximum amount of working storage in the TPF system is dependent upon the available amount of main storage minus the amount of fixed storage. The amount of working storage available to application programs depends on the number of system control blocks, common blocks, and other system resources in use at any given time.

Fixed Storage

Fixed storage is permanently allocated for specific purposes by the control program. It is not available as a system resource for use by an application program (ECB-controlled program).

Working Storage

Working storage is divided into distinct block types: I/O blocks (IOB), system work blocks (SWB), entry control blocks (ECB), 4K frames, and 4K common frames. These block types are referred to as physical storage blocks.

Application programs can request *temporary storage* in the form of logical storage blocks, which are of fixed sizes: 128, 381, 1055, and 4095 bytes.

4K Frames and 4K Common Frames: 4K frames and 4K common frames are used by the control program to satisfy requests from ECB-controlled programs for storage in a standard block size (128, 381, 1055, or 4K). The control program manages the allocation of 4K frames and 4K common frames.

A 4K frame is a 4KB block of virtual storage; logical storage blocks are allocated to an ECB from this physical block. These storage blocks are private to an ECB.

A 4K common frame is a 4KB block of virtual storage. When an Entry requests a common block, it is allocated from this physical block regardless of its size. These blocks can be accessed by all ECBs now active in the central processing complex (CPC).

Logical Storage Blocks: Logical storage blocks are used for data records, message blocks, keypoint records, and program segments. These blocks can be private to an ECB or shared between ECBs depending on whether they are allocated from a 4K frame or from a 4K common frame. When logical storage blocks are located in an area private to the ECB (that is, a 4K frame), they are known as private logical blocks or just logical blocks. When logical storage blocks are located in 4K common frames, they are known as common blocks and can be used to pass information among ECBs. See Figure 21 on page 73.

The relationship of a frame to (private) logical blocks is:

One 4K frame	= one 4KB logical block
	= three 1055-byte logical blocks
	= ten 381-byte logical blocks
	= ten 128-byte logical blocks

The relationship of a common frame to common blocks is:

One 4K common frame	=	one 4KB common block
	=	one 1055-byte common block
	=	one 381-byte common block
	=	one 128-byte common block

Notice that it is less economical spacewise to use common blocks than logical blocks. Furthermore, there is no entry protection for common blocks; these blocks are accessible by all Entries (in a CPC) and therefore risk damage by any Entry.

When an application program requests a logical storage block, the control program service routine puts the block address into one of the 16 slots in the ECB called core block reference words (CBRWs). Each CBRW is associated with a data level identified in macros as Dn (for data level *n*) where *n* is a hexadecimal number between X'0' and X'F'. For example, D3 is known as *data level three*.

Types of Dynamically Allocated Storage Available to an Application

An application program can use:

- As much as 1 MB of storage from the ECB private area (EPA) and 4 KB common frames.
- A variable amount of heap, system heap, and ISO-C stack storage. Heap, system heap, and stack storage are individually controlled by the user-defined values set in keypoint A. An application program can also request temporary storage from ECB heap storage.

  The application can also allocate variable length storage from the system heap. Unlike heap storage, system heap storage is not returned to the system when the ECB exits; the system heap can be viewed as *persistent storage*. The system heap storage, in this case, is persistent across ECBs but temporary from the system point of view in that it can be released.

See Figure 21 on page 73 for a sample virtual storage layout.

ECB Private Area (EPA)

The *get core* macro (GETCC) dispenses a logical storage block in a standard size (128, 381, 1055, or 4K). When the Entry requests a (private) logical block, it is dispensed from the ECB private area, whereas when the Entry requests a common block, it is dispensed from a 4K common frame.

The ECB private area is a minimum of one 4K frame plus the size of an ECB currently defined as 12K. The actual size of the ECB private area depends on the storage resource requests by an entry up to a maximum of 1MB.

The ECB private area is private to an Entry; that is, it is not accessible by other Entries. Regardless of its actual location in main storage, an Entry views its EPA as if it is located below 16MB.

Types of Heap Storage

Heap The *reserve a storage block* macro (MALOC) obtains a variable-sized, doubleword-aligned block from heap-resident storage. Requests from the MALOC macro are dispensed as a contiguous area of storage of variable size and are specified by the application. Heap storage is private to the ECB.

ISO-C Stack Stack heap storage is acquired by the TPF system. ISO-C Stack storage is private to the ECB.

System Heap

The GSYSC macro (or `gsysc` function) obtains a variable-sized contiguous storage area from the system heap. The system heap is not private to the ECB.

Heap storage cannot be used as logical storage blocks. This means that heap storage cannot be used as I/O buffers for I/O macros such as FINDC and FILEC. However, heap storage can be used as an I/O buffer for macros such as FILNC and tape write.

Dispatching (CPU Loop List Processing)

Dispatch control lists (DCL) are the implementation of the CPU loop lists, which were introduced in “CPU Loop (Dispatching Work)” on page 61. You may want to review that section before proceeding.

The dispatch control lists (CPU loop lists) are listed here for reference in the order that they are checked by the CPU loop:

- Cross list
- Ready list
- Input list
- Deferred list.

Items are added to the bottom of a dispatch control list by various system programs. (There is a list of control program components that place items on the dispatch control lists in “CPU Loop (Dispatching Work)” on page 61.) Although the dispatch control lists are processed by the CPU loop in a specific order, which therefore implies an order of processing priority, there is still another level of priority involved. Without this additional level of priority, items would always be added to the bottom of a list. By design, some items associated with control program services are given priority privileges and are placed at the top of the list.

Items are deleted from the top of a dispatch control list only by the CPU loop when it dispatches work. Therefore, items are added by various system processes but are removed only by the CPU loop process.

There is a set of dispatch control lists for each I-stream engine in a central processing complex (CPC) that are referenced through Page 0, therefore eliminating the necessity (and overhead) of processor locking except when processing the cross list (“Switching an Entry to Another I-Stream Engine” on page 86 describes cross list processing).

Dispatch Control List (CPU Loop List) Management

A dispatch control list (DCL), as an implementation of a CPU loop list, is formally a queue or waiting line with a top and bottom. Work is always dispatched (removed) from the top of a list and the succeeding items (more work) are promoted from within the list.

In a DCL item, the fields that are used to manage a DCL are: the block address and the post-interrupt (PI) vector. The block address is the address of a working storage block upon which the program identified by the PI vector operates.

Here is an example to show how work begins in the TPF system.

- When all of the I/O operations have been processed on behalf of an Entry waiting for I/O to complete, an item is placed at the bottom of the ready list by the interrupt handling programs. Processing associated with this I/O is shown in Figure 22 on page 78.
  - The block address contains the address of an I/O block (IOB) created during I/O processing, which in turn points to the ECB of the Entry for which the I/O was performed.
  - The post-interrupt (PI) vector points to an I/O post-interrupt service routine that will be invoked when the item is removed from the top of the ready list, which restores the Entry's registers that were previously saved in the ECB.
- A pointer in the ECB locates the data handled by the I/O request.
- Thereafter, the Entry is invoked at the point where it was previously interrupted because of a wait condition. This point of return was also saved in the ECB.

In a central processing complex (CPC) of only one I-stream engine, this I/O processing example is an accurate sketch for dispatching an Entry for more processing. The procedure is more involved in a CPC of several I-stream engines, but these basic notions still hold true.

All DCLs are circular in nature, which implies that there is a fixed number (maximum) of entries in a list. However, the TPF system uses a technique to dynamically expand a list as it is about to overlap. This is part of the flexibility required in TPF to handle those periods of high system activity.

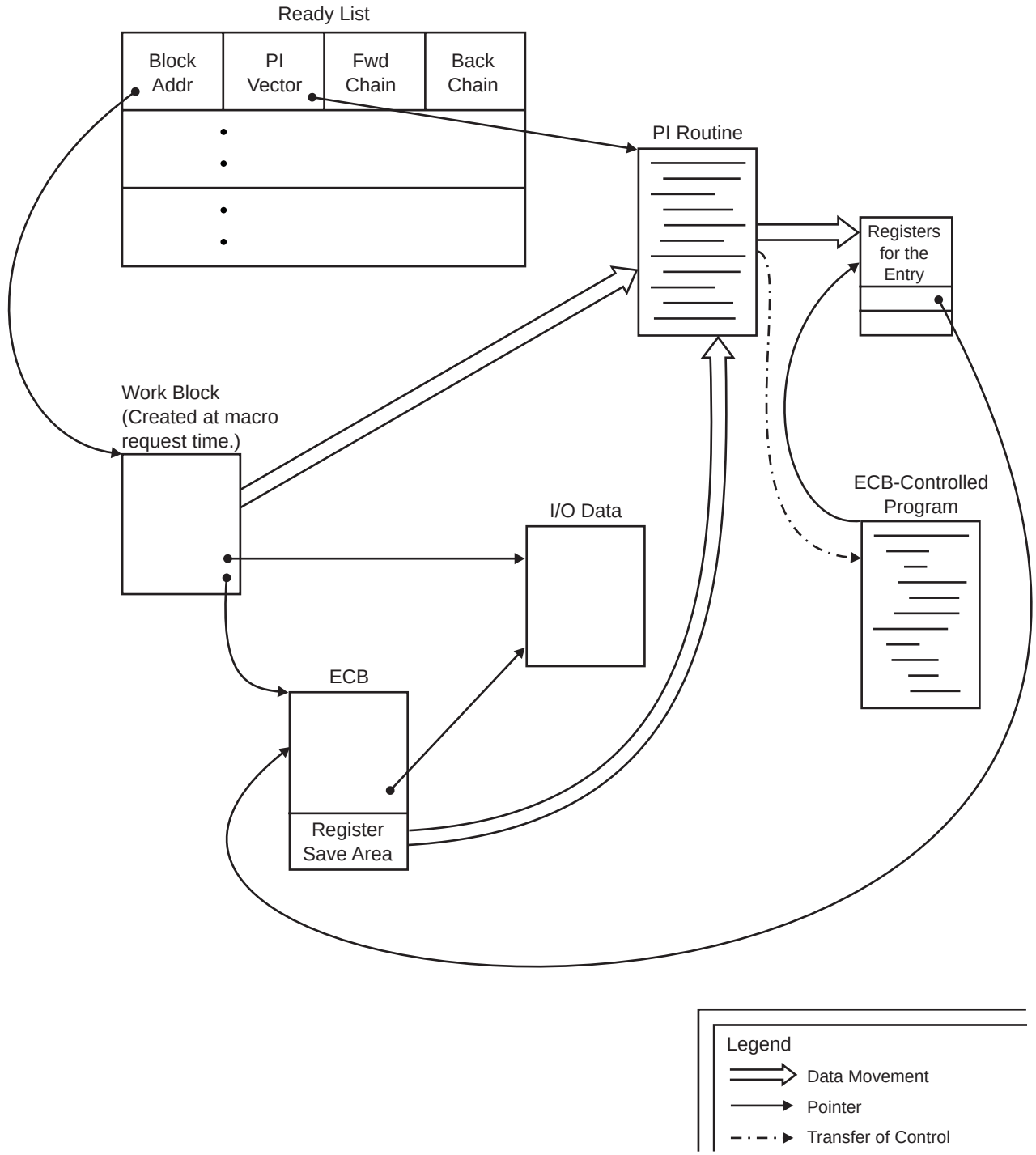


Figure 22. Action on the Ready List

Enter/Back (Program Linkage)

TPF programs are invoked through one of the following enter macros or through one of the create macros, which are described in “Create Entries with Create Macros” on page 88.

ENTNC — Enter with no return. The calling program does not expect a return of control.

ENTRC — Enter with return. The calling program expects to get control back. When a BACKC macro is encountered during Entry processing, BACKC returns control to the last program that issued an ENTRC.

ENTDC — Enter and drop previous programs. An ECB-controlled program is invoked and the Enter/Back control information that was saved in the ECB is reinitialized to remove linkages to all previous programs.

SWISC TYPE=ENTER — Transfer the ECB to another I-stream engine and drop previous programs. This macro performs the function of ENTDC while transferring the ECB to another I-stream engine.

When an ISO-C dynamic load module (DLM) issues external function calls, a stub is processed that provides the bridge for an ISO-C program into the TPF enter/back services. A DLM is a load module that contains multiple object files linked into 1 load module with 1 entry point.

The code expansion of the enter macro contains a unique external symbol (external in the sense of the MVS assembly process). The TPF linkage editor (LEDT) uses this external symbol to create the linkage to reach the appropriate system service routine.

TPF loaders and the system allocator manage program loading and associating program names with system storage.

- The allocator uses program allocation information specified during system generation to create the offline system allocator table and the online program allocation table
- The system loader (known as the general file loader) is a two-part process: offline load and online load.
  - The offline loader builds a loader general file with help from the TPF linkage editor (LEDT, or for ISO-C, Nova LEDT). The linkage editor uses the data in the SAL table to resolve the external symbols (VCONS to the MVS assembly program) that refer to application program names. All external references are resolved before application programs are placed at system resident locations.
  - The online loader (ACPL) uses the loader general file to place the online programs in their designated locations. The online loader is the mechanism used by the TPF system to build the production TPF environment.

The system service routine responsible for handling Enter requests:

- Knows that the program is already in main storage, or
- Detects that the program is being accessed from file storage because of a previous reference, or
- Detects that the program must be accessed from file storage.

If the program is already in main storage, the TPF system also knows where in main storage the program is located. If a program must be accessed from DASD storage, the TPF system obtains the necessary space in main storage to hold the program. This (main) storage management is transparent to the application program.

In summary, the system loader and system allocator represent the procedure that *catalogs* the application programs to their system residence. The mechanism that moves file-resident programs from file storage to main storage is system code that

is reached as a result of an Enter macro request. BACKC implies that the entered program is returning control to a calling program. A distinction between Enter and Entry is useful:

Entry
: A process (possibly consisting of many programs) for which a separate ECB was created. An Entry is usually created as a result of an input message. The system can also create Entries that are not the direct result of an input message.

Enter
: The act, through the use of an Enter macro request, of invoking (transferring control to) another program. The entered (called) program will use the ***same*** ECB as the entering (calling) program. If the called program must be moved from file storage to main storage, the TPF system obtains the storage and brings about the movement. Both the entering (calling) and entered (called) programs are part of the ***same*** Entry. As programs return, main storage blocks can be released by the system through the use of the BACKC macro.

Program Nesting

Several related programs can be used to process a single message under control of a single ECB. A chain of enter-with-return requests (ENTRC macro), all under control of the same ECB, is called *nesting*. A program nesting area is used to bring about online program linkage that correlates the ECB to the programs that it is referencing through enter-with-return macro requests.

There is a program nesting area within the ECB to hold Enter/Back linkages. The program nesting area within the ECB is a fixed size, thereby limiting the number of programs that can be nested. At system generation time, you have the option of accepting the limitation (which is designed to be sufficient for the average application) or specifying unlimited nesting.

When an application program issues the ENTDC macro, all items in the program nesting area are cleared. ENTDC causes an ECB-controlled program segment to be invoked and all previous Enter/Back information, saved in the nesting area, is deleted from the ECB.

TPF System Program Classifications

Several types of program classifications are used in the TPF system. These program classifications are associated with Entry management and the storage residence of programs during the online operation of the TPF system.

Control Program

Programs that are included in the online environment, other than ECB-controlled programs, are collectively called the control program (formerly known as the core resident control program). Although such programs reside on TPF online file storage for the purpose of system restarts, the programs are assigned to main storage to control the online execution of the TPF system. These programs are link edited together for online execution with the MVS linkage editor (rather than the TPF linkage editor).

Many programs within the control program are reentrant, in the context of simultaneous execution on several I-stream engines. This means that the program is shared among all I-stream engines and the program references data through private registers and Page 0 in the I-stream engine. Such a program can, however, enter a critical region to update data that is shared among all I-stream engines, in

which case a processor lock must be invoked when the data is updated. Nevertheless, within an I-stream engine such a reentrant program remains serially reusable; that is, the program runs to completion before being invoked by another Entry on the same I-stream engine.

In a multiple I-stream engine environment, register save areas for a reentrant control program component necessary to make nested subroutine calls are private to an I-stream engine. If save areas were held within the subroutines, as on a uniprocessor, the simultaneous use of such a component could fail unless the save areas were treated as shared data on which locking would be necessary. Such locking would cause prohibitive performance degradation, and frequently make a multiprocessing environment look like a uniprocessor. So, private save areas, also called stacks, are used in the TPF system to maintain reentrancy for control program components across multiple I-stream engines. In some operating systems, stacks are used to allow a routine to make recursive calls to itself. This cannot be done in the TPF system.

A set of pointers to the private stacks is held in Page 0 of each I-stream engine. The stacks are classified according to the control program component that uses the area; for example, the SVC save area.

For example, in the course of invoking the FINDC macro service routine, the macro decoder program initializes the SVC stack pointer (which is register 13, a convention followed by the TPF system), which permits the FINDC macro service routine to make nested subroutine calls. See Figure 23 on page 82.

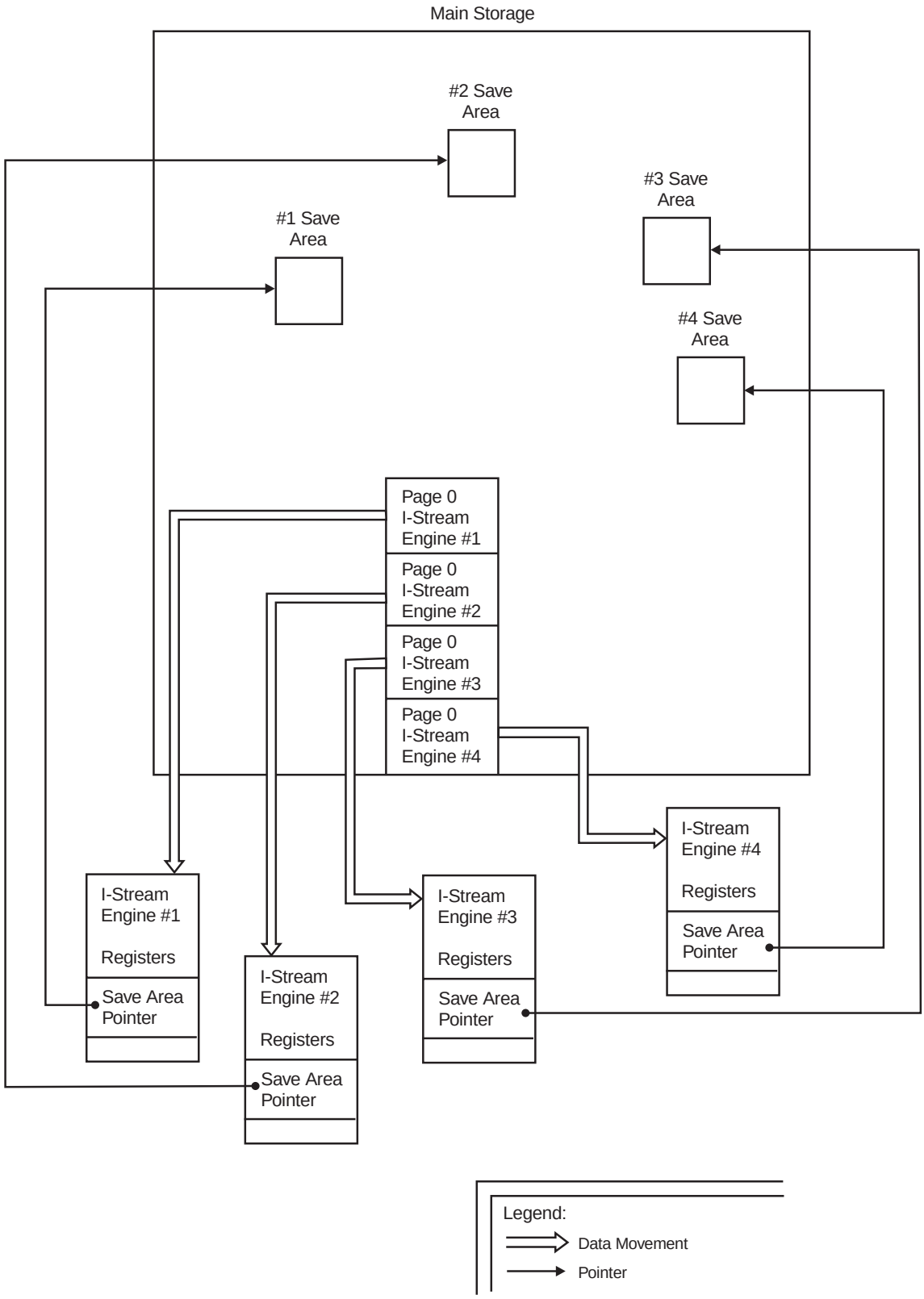


Figure 23. Reentrant Stacks

ECB-Controlled Programs

A program that uses an entry control block (ECB) for execution is called an ECB-controlled or E-type program. An ECB-controlled program must fit into a 4KB main storage block (unless it is an ISO-C program) and is invoked by an enter, create, or control transfer macro request. It is released by a back or exit macro request. (Control transfer is discussed under “TPF System Control Transfer”.)

ECB-controlled programs that are used to perform system services are called system ECB-controlled. User installation defined programs are called application ECB-controlled programs.

In summary, there are:

- Main storage resident ECB-controlled programs
- File resident ECB-controlled programs
- System ECB-controlled programs, which may be main storage or file resident
- Application ECB-controlled programs, which may be main storage or file resident.

ECB-controlled programs can be written in high-level assembler, C, or C++ programming language. The program type is identified during the offline load process.

TPF System Control Transfer

The control program normally executes in supervisor state without an ECB. However, there are situations when the control program requires work to be performed in the ECB-controlled environment. An example is when OPZERO makes the transition from the control program environment to the ECB-controlled environment to start the processing of an input message.

The control transfer macro (CXFRC) is the facility that can be issued by any routine in the control program. This facility permits several modes of operation:

- Create an ECB and place it on either the ready or the input list (that is, a CPU loop list)

  The control program routine specifies:

  – The CPU loop list on which to place the ECB, based on the priority of the work; the ready list has higher priority than the input list.
  – Where to place the ECB on the specified list, at the top or bottom of the list
  – The address where control is passed when the ECB gets to the top of the specified CPU loop list.

  Notice that after the ECB is placed on the specified list, control is returned to the control program routine. The ECB must wait until it gets to the top of the specified list before it is activated.
- Create an ECB but do not place it on a CPU loop list.

  Control is returned to the control program routine after the ECB is created. The control program routine can then perform additional processing (such as initializing work areas in the ECB) before the ECB is scheduled for processing. The control program routine is responsible for scheduling processing of the ECB by either placing it on a CPU loop list or by using an ENTNC macro request to activate it.

You may not be willing to review all of “Step 4. Create an ECB and Select an Application” on page 67. However, it is recommended that you review “Step 4b. OPZERO Creates and Initializes an ECB” on page 67 because you are now in the position to understand how a newly created Entry is invoked by the OPZERO program. Remember that OPZERO is invoked by the CPU loop when it takes an item off the top of the input list on the main I-stream engine.

Action on the Cross List (Switching I-Stream Engines)

A useful abstraction for describing the dispatching of processing work in a tightly coupled multiprocessing environment is shown in Figure 24 on page 86. This abstraction suppresses the fact that in reality, the CPU loop program comes in two versions: one version is given an affinity to the main I-stream engine and the other version is shared by all other I-stream engines. The distinction between the two versions is not necessary to describe the action taken on cross list items. In Figure 24 on page 86, I-stream engines are shown as “CPU n” because this is the way they are identified in online system tables.

The ideas emphasized in Figure 24 are:

- Multiple I-stream engines (CPUs) share main storage.
- A single program, the CPU loop in this case, in shared main storage can be executed by all of the several I-stream engines. The execution can occur simultaneously.
- Data in shared main storage, which is unique to an I-stream engine (the cross list in this case), is accessed using Page 0 referencing.
- All I-stream engines have access through pointers in the I-stream status table to the cross lists on each of the other I-stream engines. Processor locking must be invoked to protect data that is modified in the tables associated with a cross list. Notice that although there are multiple cross lists, they are shared by (that is, accessible to) all I-stream engines. A program executing in any I-stream engine can add an item to the cross list of any other I-stream engine, but only a program executing in an I-stream engine with a Page 0 reference to its own cross list is permitted to remove an item from a cross list.
- A single flow chart, Figure 18 on page 64 in particular, represents execution sequences that can occur on all I-stream engines. In following a specific instance of I-stream engine switching, the flow chart remains the same, but ***you*** must associate different I-stream engines with different paths on the flow chart.
- In a uniprocessing environment of just the main I-stream engine, the flow is still valid.

Also, remember that most of the main storage resident control program is reentrant across I-stream engines, but is only serially reusable within an I-stream engine. Keeping all of the previous ideas in mind is not always easy. After reflecting upon Figure 24, you should be ready to consider the following description of I-stream engine switching.

Conceptually, the cross lists represent a communication mechanism (mail boxes of sorts) for moving processing work among I-stream engines in a tightly coupled multiprocessing environment. Each I-stream engine must be given the location of every other cross list, a rudimentary form of routing. The cross list references are done through the I-stream status table; the pointers to this table are handled during system initialization. During IPL, each I-stream engine is assigned a number known as the I-stream engine ID. This ID is, for example, placed in a field in an ECB and is passed from one I-stream engine to another through system control blocks such

as IOBs. This permits a response to be returned to the I-stream engine on which a request was made; for example, the results of servicing a tape I/O request.

All processing work transferred to another I-stream engine must be done by means of a cross list, which reduces the amount of locking that would be necessary if all CPU loop lists could be accessed by any I-stream engine. Furthermore, multiple cross lists reduce the number of instructions needed to find an item of work for any given I-stream engine; scanning and processor IDs would be required if there was a single shared cross list.

The structure of a cross list item reveals the essence of the cross list interface. An item on a cross list has the following 4-byte fields:

Field	Definition
Parameter 1	The data in this field is particular to the post-interrupt routine that is to receive control.
Parameter 2	The data in this field is particular to the post-interrupt routine that is to receive control.
Forward pointer	The items in the cross list are forward chained.
Backward pointer	The items in the cross list are backward chained.
PI vector	This field points to the routine that is given control when the item of work reaches the top of the cross list.
I/S originator	This field identifies the I-stream engine from which the processing work originated. It is used primarily by system tracing facilities.

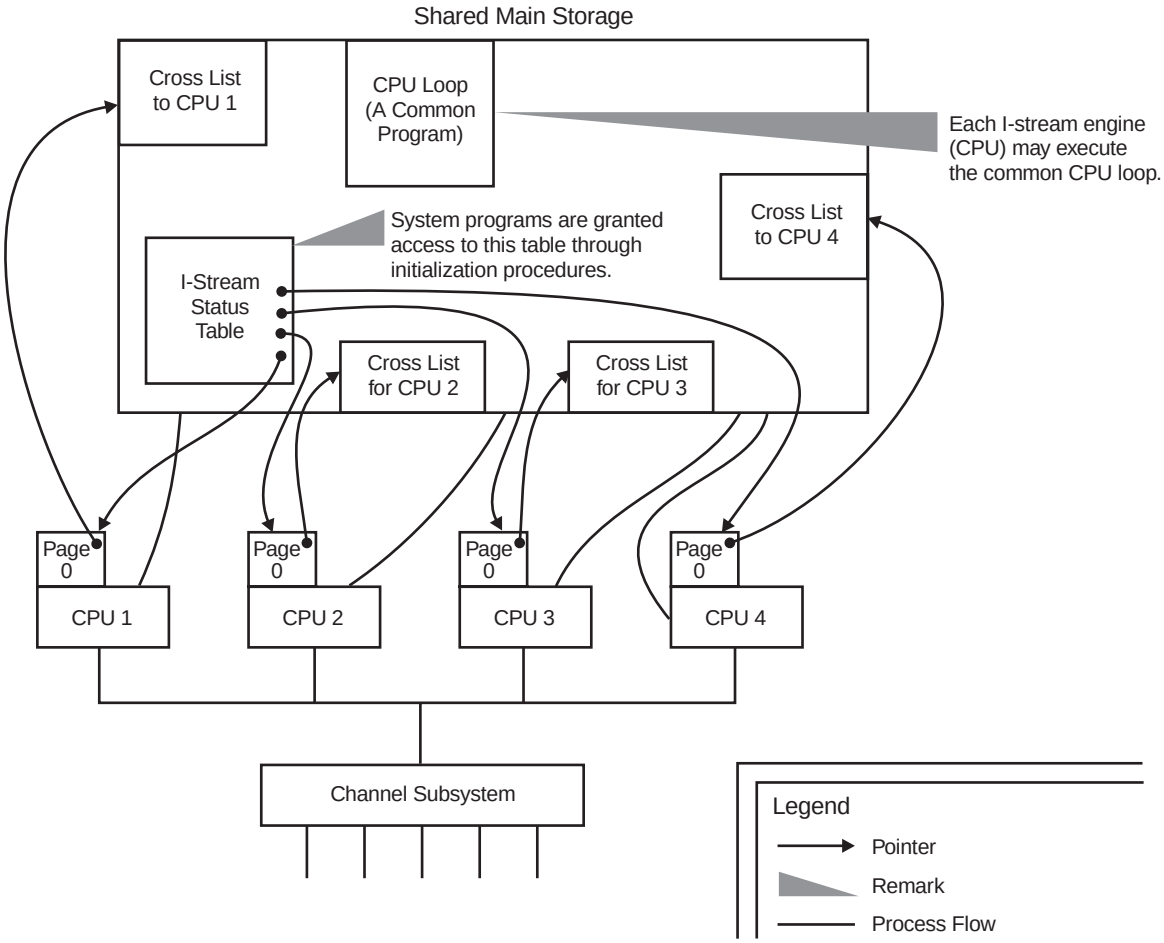


Figure 24. Multiple I-Stream Engine Processing Abstraction

Switching an Entry to Another I-Stream Engine

An outline of the procedure for moving an Entry from the main I-stream engine to an application I-stream engine is given in “Step 4c. COMM SOURCE Invokes the Application” on page 68. The detailed processing flow when the SWISC macro is executed on the main I-stream engine as well as subsequent processing follows here:

1. COMM SOURCE issues an SWISC macro request on the main I-stream engine. The SWISC macro expansion is found in the routing control application table (RCAT). The SWISC macro input parameters, in this case, include the name of the program segment to be entered and the option to allow the load balancing routine to select the I-stream engine on which the Entry should be dispatched. As a result of electing to execute the Entry on an application I-stream engine, the need arises to place a work item on the ready list of an application I-stream engine.

   In switching an Entry, the Entry is always dispatched from the ready list and not directly from the cross list. The cross list is the highest priority dispatch list and is used for control program services of relatively short duration, with the goal of keeping the list empty or short. An item is only removed from a list when the work associated with a previous item has completed or is suspended. Although

an Entry is characterized as expeditious processing, an Entry is a long-running process in comparison to control program services.

2. The SWISC service routine, executing on the main I-stream engine, uses the $ADPC macro (add to designated list in designated I-stream) to build the following parameters:
   - The address of the routine to be placed in the post-interrupt (PI) vector field of the cross list item on the application I-stream engine. This PI vector identifies a routine that will ultimately add an item to a designated list in the application I-stream engine; in this case, the ready list.
   - The ID of the I-stream engine to receive the work.
   - The storage block address to be added to the application I-stream engine's ready list; in this case, the ECB.
   - The PI vector to locate the routine to be invoked from the ready list of the application I-stream engine; in this case, the termination of the SWISC service routine. Do not confuse this with the value placed in the PI vector field of the cross list itself.
3. The $ADPC macro service routine always passes four parameters to the $CRISC (add to cross list in designated I-stream) macro service routine, which, in this case, places a four-field item on the cross list of the designated I-stream engine.
4. When the item reaches the top of the cross list on the application I-stream engine, the PI vector to $ADPC (the PI vector of the cross list) causes the terminating portion of the $ADPC routine to receive control, which places the ECB address and the PI vector to the SWISC routine on the ready list, and a return is made to the *top* of the CPU loop program.
5. The two-field item of *ECB addr* and *PI to SWISC* is now at the bottom of the application I-stream engine's ready list. When this item reaches the top of the ready list, the terminating portion of the SWISC service routine that was originally invoked on the main I-stream engine is given control. This terminating routine is the equivalent of an ENTDC (enter and drop all existing programs) macro request.

Starting an Entry on the application I-stream engine requires the use of the control transfer interface to get from OPZERO to COMM SOURCE on the main I-stream engine, the SWISC macro to place an item on the cross list of the application I-stream engine, and the ENTDC macro service routine to invoke the first segment of an application package on the application I-stream engine. Although the SWISC macro is always invoked by COMM SOURCE on the main I-stream engine, in a uniprocessing environment, the Entry is never switched but is dispatched from the ready list of the engine in which the SWISC macro is issued, because the ID of the engine to receive the work is the main I-stream engine itself. This can also happen in a multiple I-stream engine environment if, for example, all the application I-stream engines are busier than the main I-stream engine.

Switching I/O Processing Between I-Stream Engines

To uncover the processing sequence to handle an I/O request, consider the case where an Entry, executing on an application I-stream engine, uses a FINDC macro request followed by a WAITC macro request. You should now refer to Figure 18 on page 64, and keep the description provided by Figure 24 on page 86 in mind.

1. The FINDC macro service routine creates an IOB that points to the ECB of the requesting Entry and is used to control an I/O operation on behalf of the Entry.

2. If the DASD device needed to complete the request is busy, the IOB is put on the device queue. If the DASD device is not busy, the I/O is started immediately from the I-stream that issued the FINDC request.
3. The FINDC service routine, running on an application I-stream engine, restores registers with the register save area found in the active ECB. Thereafter, the old SVC PSW is loaded and processing continues in the entry on the application I-stream engine from which the FINDC I/O request was made.
4. The entry ultimately issues the WAITC macro request. If the I/O has not started (because the device queue contained other IOBs) or the I/O has not completed, the ECB will be suspended and another entry will be dispatched on the I-stream.
5. If the DASD device to which the I/O was queued was busy, the I/O will eventually be started by the I-stream that processed the last interrupt for the IOB before our FINDC request on that device.
6. Finally, the I/O will be completed and issue an I/O interrupt on any I-stream. By using the $ADPC macro, the IOB is put back on the ready list of the I-stream that initially issued the FINDC macro.
7. When the IOB reaches the top of the ready list, the FINDC post-interrupt routine is called to place a pointer in the ECB that locates the data read from disk. This routine also releases the main storage block used to hold the IOB. Assuming no other I/O is pending, a return is made to the entry that issued the FINDC macro request. If additional I/O is pending for the ECB, control is returned to the CPU loop.

Create Entries with Create Macros

A set of macros permits an active Entry to create another independent Entry. (Note that an input message is not directly involved in creating the new Entry.) The creating Entry can pass information to the created Entry. In principle, the procedure is similar to a control transfer but is done on behalf of an existing ECB-controlled program rather than on behalf of a control program component. The different types of create macro requests are:

- Create an immediate Entry (CREMC), which places the Entry on the ready list.
- Create new synchronous ECBs (CRESC) which places the Entry on the ready list.
- Create time-initiated Entry (CRETC), which places the Entry on the ready list after a specified time interval; an attached storage block can be passed to the new Entry.
- Create a low priority deferred Entry (CREXC), which places the Entry on the deferred list for low-priority deferred processing.
- Create an immediate or deferred Entry with attached storage block (CREEC), which places the Entry on the ready list or the deferred list. When the new ECB is created, information from the creating ECB is placed in a storage block attached to the new ECB.
- Create an Entry on a specified I-stream engine (SWISC TYPE=CREATE), which places the Entry on the input, ready, or deferred list on a designated I-stream engine.

Common I/O Handler (CIO)

Within the channel subsystem, the input/output processor (IOP) supervises the flow of data from shared main storage to I/O engines that, in essence, represent devices. The IOP does dynamic pathing (routing) to connect a device to an I-stream engine. The term *subchannel* is synonymous with *device* in TPF terminology; that is, there is a unique subchannel address for each device.

The TPF common I/O handler (CIO) manages I/O operations through a clearly defined macro interface that permits the set of CIO macros supporting each I/O function to make use of a centralized service structure. The I-stream engine processing related to I/O instructions, channel programs, and related I/O addressing schemes of the past are essentially untouched. CIO, however, takes advantage of the benefits of the ESA channel subsystem such as dynamic pathing.

File Storage (DASD) Accessing

Although several different file storage devices (DASD), each with their own unique characteristics, are supported by the TPF system, all file accessing is conceptually done in a similar fashion. The generic terms used for file accessing are find (read a record) and file (write a record).

There are many variations of find and file, such as *file the record but do not release the working storage block*, and *find the record and hold (reserving updating exclusively for the holding Entry)*.

WAITC is an important macro used with I/O operations. WAITC is an application program request to delay further processing until all the pending I/O operations for the application program issuing the WAITC are completed. An I/O counter within the ECB is used for this control. WAITC can be implicitly used with find and file macro requests; for example, *find a record and wait*. While data transfer occurs, the TPF system is free to shift control to other Entries that are ready for processing.

A level is one of the 16 pairs of data fields in the ECB. These fields are:
- File address reference word (FARW)
- Core block reference word (CBRW).
- File address extension words (FAXWs)
- Detailed error indicator.

The file address reference word (FARW) is used to pass a file address between the application and the control program. The core block reference word (CBRW) is used to pass an address of a main storage block used for storing data. A main storage block is automatically obtained by the control program for a find macro request. Generally, the main storage block is released upon completion of a file macro request. However, the EXITC macro, summarized in “Entry Termination (EXIT Processing)” on page 92, releases all main storage blocks associated with an Entry.

The allocation of data to physical file storage and the use of find and file macros have implications that are described in more detail in Data Organization. The important aspects are:

- The record sizes used with find and file macros correspond to the same sizes used by the main storage allocation functions.
- Predefined records, such as indexes to more dynamic data, are managed in a file area known as *fixed-file records*. A set of data within the fixed file area is identified as a record type. Within a fixed-record type each record is identified

with an ordinal number. An application cannot create a new record type during execution. As a matter of fact, all fixed record types, including the number of records they contain, are defined and allocated through the use of an offline program during system generation. Allocation means the identification, through system tables, of the correspondence between physical devices and the data structure an application program is permitted to use.

- The dynamic requirements for file storage are satisfied by areas known as pools. The allocation of the pool record area to physical devices is also done during system generation. However, the pool record area represents a large repository of file storage accessible by all applications. The records within the pool area are ***dispensed*** on an ***as needed*** basis. The TPF system maintains an availability bit indicator for each pool record. When an application requests pool file space, the TPF system returns the address of one of the available records in the pool. When the application is finished with the record, the pool record associated with the address is marked available.

The TPF system handles the details of obtaining the physical location of data, that is, addresses that are directly utilized by the hardware. This is done in stages:

- An application request to find or file a record must be preceded by one of two functions:
  - Use of the get file storage macro to obtain the address reference of a pool record.
  - Use of a system program to obtain the address reference of a fixed record, given the record type and ordinal number.
- An application program, in turn, passes a file address reference to I/O service routines through the use of FIND and FILE macros and the FARW of the ECB.

The file address information is in a format for either pool or fixed data record references by the time a find or file macro request is issued. The file address used by an application program is sometimes called *symbolic* to emphasize that the translation required to obtain the physical location of the data is not complete.

The use of data within fixed records as indexes to pool records is a common technique used in the TPF system environment. The combination of fixed and pool structures provides the application designer with the tools to implement application-oriented data management subsystems.

The TPF Database Facility (TPFDF) product provides a high-level application program interface (API) for database organization and accessing. See “TPF Database Facility (TPFDF)” on page 149 for additional information.

TPF System Magnetic Tape Support

Magnetic tapes are used for online and offline processing and can be used for both input and output in the TPF system. Tapes are categorized as either real-time or general tapes.

Real-time tapes are used to log transactions and collect dynamic system information for use by TPF utility and performance reporting programs and to record main storage dumps resulting from system errors. Real-time tapes are available to any Entry in the system at any time, but ECB-controlled programs that use real-time tapes cannot depend on creating consecutive records. Offline programs that eventually process real-time tapes contain logic to select specific records. The TPF system uses real-time tapes only for output.

Like most operating systems, the TPF system provides the customary tape open, close, read, and write macros, which are used when accessing a general tape. These macros are adequate for low priority jobs using a sequential data structure. Some TPF online batch-oriented Entries with excessive running time are segmented into several shorter entries, called phases, through the use of create entry macro requests. General tapes are usually associated with an entire transaction (in this case, *transaction* very closely corresponds to a batch-oriented *job*). TPF tape reserve and assign macro requests permit a transaction to pass an open set of general tapes from one Entry to another Entry.

Symbolic addressing is used for both real-time and general tapes, thus permitting application programs to address tapes through symbolic tape names assigned to hardware addresses. For example, the primary real-time tape is designated as RTA while the logging real-time tape is designated as RTL.

The tape status table (TSTB) is used to control tape operations. This table contains hardware addresses, device status, symbolic name assignments, and queuing and chaining information.

Tape devices are, in general, unique to a single central processing complex (CPC) in a loosely coupled complex. Magnetic tape addresses are allocated to a particular CPC through the use of a shared pool of addresses assigned as a result of an operational message that manages tape configurations.

Unit Record Support

Unit record equipment can also be used by low priority online batch-oriented Entries (such as report generators) that require printers or card readers as input/output (I/O) devices. If indiscriminately activated, Entries utilizing unit record equipment can impact the performance of the TPF system. Priority *job* tables in the Unit Record Message Editor program permit low priority unit record Entries to be activated only when a predefined number of main storage blocks are available. Higher priority Entries can be activated any time; therefore operator discretion is needed. An abort procedure is provided but requires a rerun of the Entry because no restart mechanism is provided.

Like general tapes, unit record devices are assigned to Entries. This ensures consecutive printing and card reading. However, unlike general tapes, no *reserve* unit record function is provided to pass control of unit record devices to another Entry for further processing.

Unit record device addresses are assigned to a particular CPC.

Console Operations

Commands represent the operational command language of the TPF system. This language provides the system operator with facilities to monitor activity in the system and levels of system resources, and to exercise some control over the system. Obviously, this is the interface between the outside world and the data maintained by the system in the management of its resources.

Commands can be entered from specially designated *command* terminals and workstations, called computer room agent sets (CRAS terminals), that are attached through communication facilities. CRAS terminals are used in the operation of the TPF system ***in addition*** to the hardware consoles that are normally used by an installation's operations staff.

The IBM Extended Operations Console Facility/2 (EOCF/2) system provides enhancements to console operations, such as the capability for automation and the ability to control and monitor multiple TPF host systems from a single workstation.

Error Recovery

Errors can occur at any point during TPF operations. There can be programming errors, such as incorrect macro parameters, or hardware malfunctions, or a variety of unusual conditions. The levels of error are:

1. Hardware malfunctions that are overcome by retrying the I/O operation. In this case, error statistics are recorded but the Entry is protected from the problem (for example, unit checks).
2. An error detected by the system from which the programs related to an Entry may be able to recover. In this case, the ECB-controlled program regains control (for example, a record identification check).
3. An error detected by the system from which the Entry cannot recover. In this case, the Entry is forced to exit (for example, an addressing exception generated by an ECB-controlled program).
4. An error detected by the system that makes continued operation of the system inadvisable. This is called a catastrophic error; for example, an operation exception (programming error) within the control program. Such a failure is detected and handled by various components of the control program and may require a system restart.

The basic objectives for TPF system error processing are:

- Save as much information as possible for the technical staff's analysis and action
- Notify the central site that the error has occurred
- Respond to the program in control at the time of the error, if possible
- Determine if operation of the system should continue.

  If operation of the system is to continue, it must be restarted as quickly as possible.

Entry Termination (EXIT Processing)

An exit macro request (EXITC) is used to remove an Entry from the TPF system. System resources held by the exiting Entry are returned to the system for use by other Entries. Control is transferred to the exit program either because an exit macro was issued or because of the occurrence of a system error.

The exit program accomplishes the following:

- Disconnects from the ECB all programs associated with the Entry
- Releases all main storage blocks, identified in CBRWs, used by the Entry
- Releases all data records held by the Entry. A *held* data record means the Entry was using the system conventions to prevent other Entries from modifying the record. The system considers data records held at exit time to be an error condition and sends an error message to the computer console.
- Closes any tapes or unit record devices opened and assigned to the Entry at the time the EXITC was issued
- Ensures that all I/O initiated by the Entry is completed
- Releases the ECB.

TPF System Structural Characteristics Summary

This chapter is, in effect, a detailed introduction to the TPF system resource management facilities, done in the context of tracing the flow of a message through the system. Data Organization and Data Communications, provide additional important structural details.

An operational system is a combination of the TPF system, application programs, and people. People assign purpose to the system and use the system. The creation of an operational system depends upon three interrelated concepts:

- System definition, which is the necessary application and TPF system knowledge required to select the hardware configuration and related values used by the TPF system software.
- System initialization, which is the process of creating the TPF system tables and configuration-dependent system software.
- System restart and switchover, which are the procedures used by the TPF system software to ready the configuration for online use.

System generation and system initialization are collectively called system generation. System definition is a design phase when installing a TPF system.

System restart is the component that uses the results of a system generation to place the system in a condition to process real-time input. The initial startup is a special case of restart and, for this reason, system restart is usually called *initial program load* or *IPL*. System restart uses values found in tables set up during system generation and altered during the online execution of the system. A switchover implies shifting the processing load to a different ESA configuration. A restart or switchover may be necessary for either a detected hardware failure, a detected software failure, or operator option.

Data Organization

The organization of data is important to the TPF system. The performance objectives of the TPF system must be achieved while, regardless of the actual physical database structure, allowing the applications to access data as simple, logical structures. In meeting these objectives, the TPF system assumes the responsibility of allocating sets of data to physical devices and for converting data references to physical locations.

Some of the factors, that must be considered with regard to data organization in the TPF system are:

- Coordination of the updates to any specific record by:
  - Multiple I-stream engines in a central processing complex (CPC)
  - Multiple CPCs in a loosely coupled complex
- The ability to expand the capacity of the database by adding module (sometimes referred to as DASD) hardware
- The ability to take advantage of new module technology by not requiring the database to be all one type of module hardware
- The ability to change the physical layout of modules based on changes required by application design; that is, the location of fixed and pool files can be changed if necessary.

This chapter introduces the strategy and techniques used by the TPF system for data organization in the following areas:

- Data located on modules
- Data held in main storage
  - Virtual file access (VFA)
  - Global areas.

Database Overview

A useful way to show data organization in the TPF system is to look at an example of a database. Refer to Figure 25 on page 96 as you read through the following text.

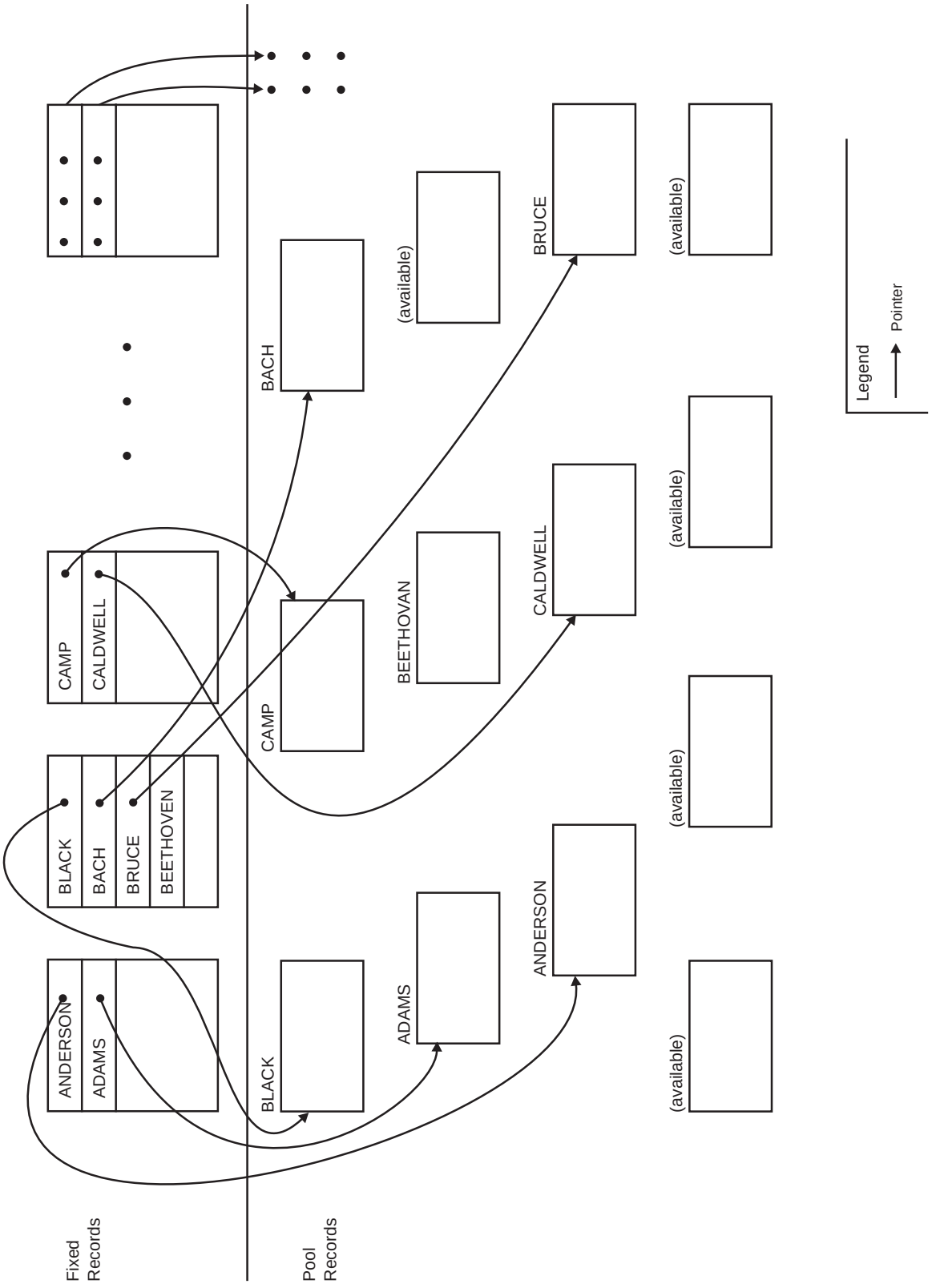


Figure 25. Database Example. Using Fixed and Pool Records.

The example involves the information that is stored pertaining to individual passengers in an airline's reservation system. This information must be organized in

such a way so as to achieve the performance objectives by minimizing the number of input/output (I/O) operations, and at the same time making the data easily accessible to the application.

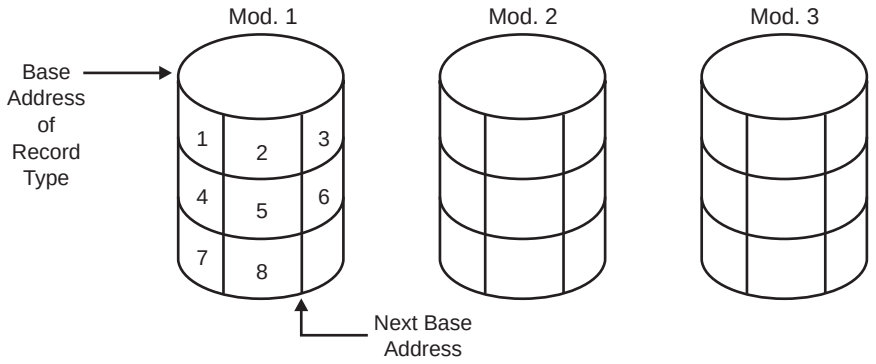
In the TPF system, a typical way to organize this data is through a hierarchical structure where the higher level is an index to the lower level that contains the detailed information about each passenger. In this example, the index is organized by the first character of a passenger's name.

The index requires a record for each letter of the alphabet. In the TPF system, fixed file records are designed to be used when the required number of records can be determined in advance (that is, when a TPF system is being designed and generated). It is desirable to be able to specify the exact number, but usually a close approximation is sufficient.

Because the number of passengers is variable, the required number of records cannot be predicted with any degree of accuracy or certainty. In addition, the data stored about those passengers can change frequently, so the records must be readily accessible. In the TPF system, pool records are designed to be dispensed on an as-needed basis; that is, an application ***asks*** for a pool record when needed and the TPF system supplies the file address of such a record. (Observe that it is the responsibility of the application to save the address of the pool record in the appropriate index record.) Once passengers have completed their journey, their detailed information is no longer required, and the accompanying pool record is returned to the system for reuse by other Entries.

Because the requests for data about passengers are random and these requests are not related to each other, the physical records on a database in a TPF system are allocated in such a way as to enhance performance. This is accomplished by placing logically adjacent records (such as, B follows A, C follows B) on different physical devices. Figure 26 on page 98 demonstrates the concept of *horizontal record allocation*, and contrasts it with the more conventional *vertical record allocation.*

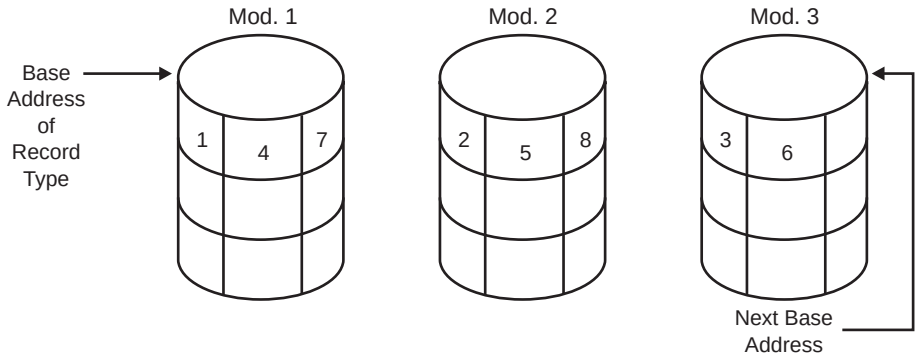
By allowing multiple Entries to access logically adjacent (sequential) records without incurring time delays because different physical devices are accessed independently of one another, the TPF system is able to meet its performance objectives.



Vertical Allocation:

Allocate a record type of 8 ordinal numbers to successive locations on a module

- The numbers represent the meaning of the next record within the record type. The meaning must be converted into physical characteristics, that is, module, cylinder, head, and record number.
- This allocation strategy is not used by the TPF system for the online database.



Horizontal Allocation:

Allocate a record type of 8 ordinal numbers to successive locations across all modules

- If each module represents an independent path to a CPU, then the accessibility to different records within the record type is improved.
- This represents the basic strategy used in the TPF system for the online database.

Figure 26. Record Allocation. Horizontal Allocation and Vertical Allocation.

Multiple Database Function (MDBF) Overview

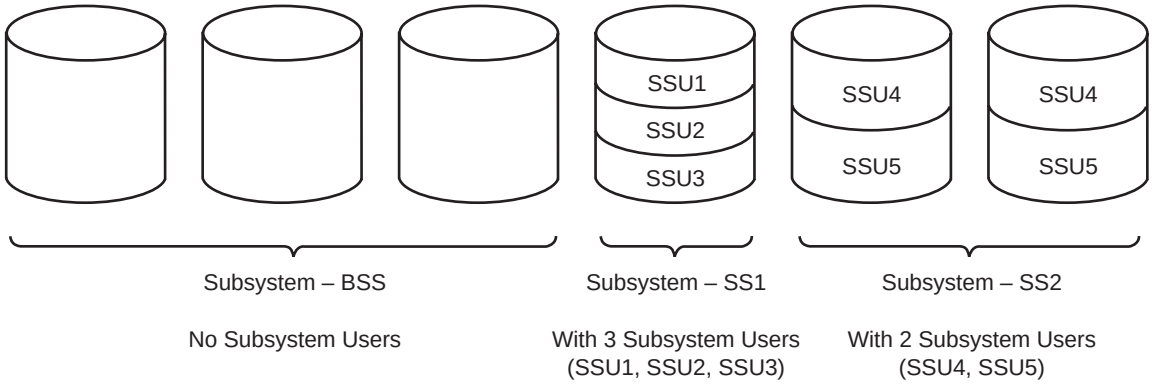
The multiple database function (MDBF) of the High Performance Option (HPO) feature provides the user with the ability to:

- Physically separate sets of data
- Logically separate sets of data
- Physically and logically separate sets of data.

The use of MDBF introduces the concepts of subsystems (physical separation of data) and subsystem users (logical separation of data). Figure 27 on page 99 shows these concepts.

The reasons for separation are those that the user installation requires, and do not necessarily contribute to meeting the performance objectives of the TPF system. An example of separating data files is an enterprise that provides a reservation service

to multiple hotels; the enterprise can choose whether to separate files of data related to one hotel physically or logically from those of another hotel.



Note:
- Subsystem provides a physical separation of data.
- Subsystem User provides a logical separation of data within a subsystem.

Figure 27. Multiple Database Function (MDBF)

Fixed Records

Fixed records form a static repository of data records, such as indexes to more dynamic data. A set of records within a fixed record area is identified as a fixed record type; a record type is the name of a set of records. Within a fixed record type, each record is identified with a record type ordinal number; the ordinal numbers are viewed logically, by an application, as sequential record numbers but are not sequentially allocated to physical storage. A fixed record type generally implies a set of data related to a specific set of application program segments.

Although the content of the records named by a fixed record type may be dynamically altered online, an application cannot create a new fixed record type during execution. As a matter of fact, all fixed record types, including the number of records they contain, are defined and placed online through the use of offline programs. The fixed record types are allocated during system generation in a manner to favor performance.

Pool Records

The dynamic requirements for file storage are satisfied by the use of areas on modules known as *pools*. The allocation of a pool record area to physical devices is done by offline programs. Several pool record types are defined by the system, based upon record attributes in order to allocate pool areas on modules. Record attributes are described in “Data Record Attributes” on page 102.

Moreover, each record within a pool record type is ***dispensed*** on an ***as needed*** basis. This means the availability status (either available or unavailable) of each record within a pool record type is managed by system service routines within the TPF system, through the use of pool directories. (Note that no such management is performed on records within a fixed record type; in essence, the system views the fixed records as being dispensed during the system generation process.)

Use Fixed Records and Pool Records

Through the use of fixed and pool records, unique application data structures are created during the design stage of an application. This implies that the structure of the data associated with an application is predefined and not readily changed once the application is implemented.

The fixed record area is an area of file storage that contains records whose symbolic addresses can be calculated using a record type or record ID and an ordinal number. The record type or record ID identifies a set of data within the fixed record area, and the ordinal number identifies a specific record within the record type.

As an example, consider again the example of organizing passengers for an airline's reservation application. Figure 28 on page 101 shows the concept of record type (in this case, #INDEX) and ordinal numbers (0 through 25).

In order to retrieve (that is, to find or read) a pool record, the application must ***know*** its address. The most common technique used by applications is to form data structures that use fixed records as indexes to pool records.

Refer again to Figure 28. The index (a fixed file record) is retrieved by using the first letter of the passenger's last name to calculate the address of the appropriate index record. The index is then scanned for the passenger's name, and, when it is found, the associated pool address is used to locate the passenger name record. This allows the selection of one of a vast number of pool records with a minimum of file accesses (that is, I/O operations).

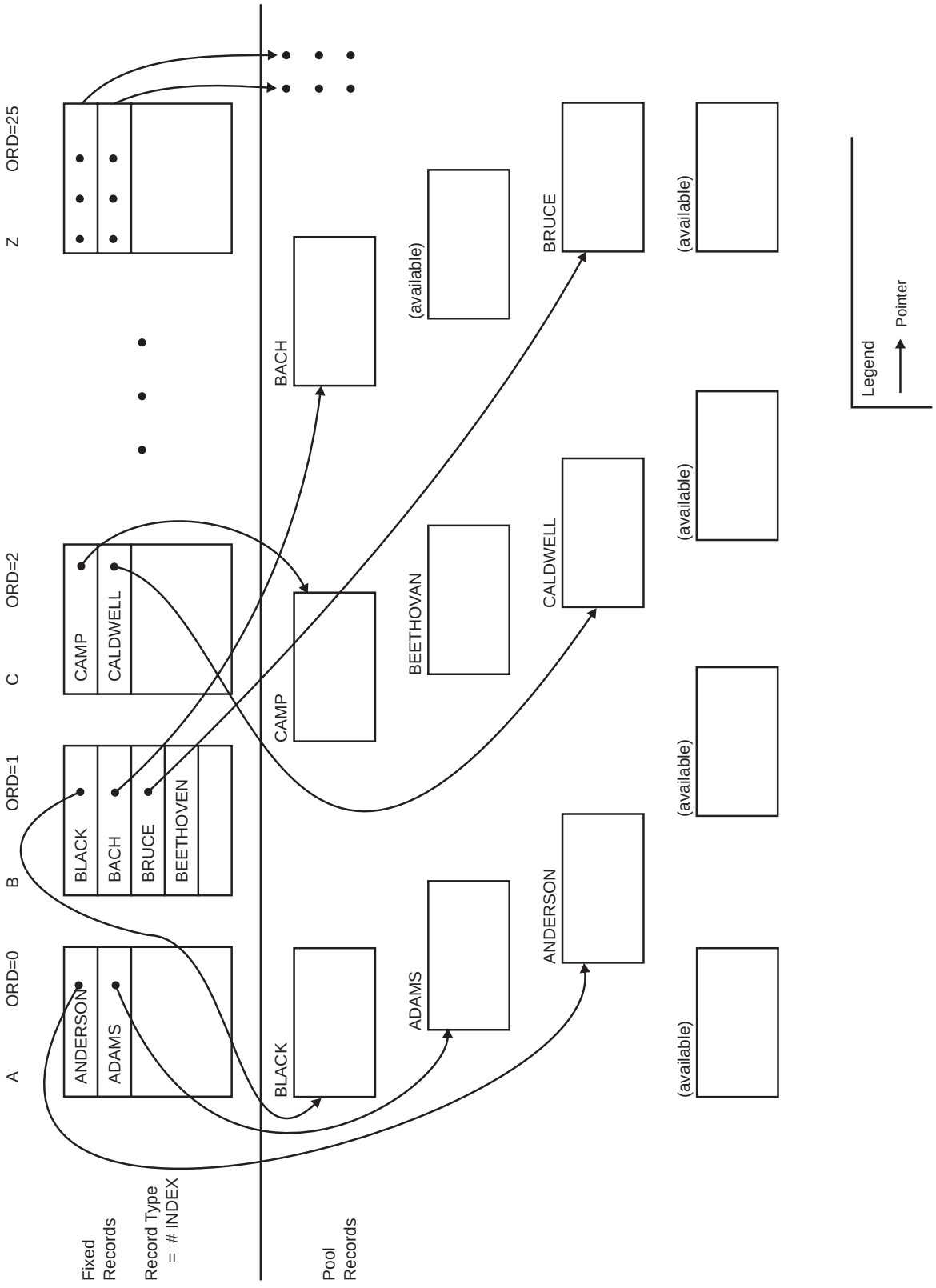


Figure 28. Record Type and Ordinal Numbers

Data Record Attributes

Data record attributes are necessary to describe data organization in the TPF system.

Physical Residence

In general, the physical location of a record is transparent to the application environment. The TPF system manages the following physical forms of storage on which data records can reside:

- Modules
- Magnetic tape
- Main storage.

Logical Device Type

The TPF system provides the capability for the user installation to use up to four different module device types within a database configuration.

The logical device types, called DEVA, DEVB, DEVC, and DEVD, are assigned to specific physical device types at system generation. The physical characteristics of the various types of devices are stored in the module file status table (MFST) and are used when the file reference of a data record is converted (mapped) into its physical hardware address.

Having access to these characteristics provides the ability to change the physical configuration of a user installation's database for the following reasons:

- To improve performance by spreading records across more physical modules so that more Entries can be accessing a set of records simultaneously
- To increase database capacity by increasing the number of physical module to store more data
- To improve performance by taking advantage of the latest module technology while maintaining whatever technology is currently in use by the user installation.

Record Size

In the TPF system, the logical size of data records is predefined. The overhead of managing data is made more efficient by only allowing a limited variety of record sizes.

The logical size of a record within the TPF system is always one of three sizes:

Small (381 bytes)
Large (1055 bytes)
4K (4095 bytes)

Record Duplication

In the TPF system, data records (on modules) can be duplicated, which means that there are two copies of a data record on the database. The copies are referred to as the primary record and the backup record, sometimes called the duplicate record or dupe.

The record allocation procedure assigns each copy of a duplicated record to different modules in order to provide an alternate copy of the record in the event of system errors (hardware or software). Duplicate records can also provide parallel paths for finding (reading) a record in a TPF system that is not loosely coupled.

The primary and backup records must be allocated to the same logical device type. Both fixed records and pool records can be duplicated.

Record Longevity

By definition, fixed records are maintained by the TPF system forever while pool records are maintained for an interval of time, which is determined by the application environment (in terms of seconds, hours, days, weeks, or months).

The user installation actually makes the final definition between a short time and a longer time for pool records. Generally speaking, a short time is usually thought in terms of seconds, minutes, and hours. Two intervals of intended use for pool records are supported:

Short-term pool records
Short-term pool records are intended for data that exists for a relatively short time. The system considers a short-term interval to be no longer than the time required to complete a transaction (see “Transaction Defined” on page 11).

Long-term pool records
Long-term pool records are intended to be used by programs that require the data to exist much longer than the life of a transaction. This may be an indefinite period of time.

Pool Record Types

Pool records are classified according to the attributes of size, duplication, longevity, and residence (type of device). The basic pool types are:

- Small short-term pool (SSTx)
- Small long-term (not duplicated) pool (SLTx)
- Small long-term duplicated pool (SDPx)
- Large short-term pool (LSTx)
- Large long-term (not duplicated) pool (LLTx)
- Large long-term duplicated pool (LDPx)
- 4K short-term pool (4STx)
- 4K long-term (not duplicated) pool (4LTx)
- 4K long-term duplicated pool (4DPx)
- 4K long-term duplicated FARF6 pool (4D6)

where x is A, B, C, or D and represents the type of device defined for that pool type at system generation time.

Types of Fixed File Records

Fixed file records can be classified according to the following sizes:

- Small (381 bytes)
- Large (1055 bytes)
- 4K (4095 bytes).

Do not confuse types of fixed file records with the concept of fixed file record types, which are names of sets of fixed file records.

Record IDs

Record IDs are used within the TPF system to ensure and validate database integrity. Every record in the system, whether fixed or pool, must be associated with a two-byte record ID.

A record ID is placed in each fixed record when the database is initialized, and in each pool record by application programs as these records are acquired.

This record ID is given as a parameter within a file address reference word (FARW) of the entry control block (ECB) when a record is accessed with a find or file type macro request. Database integrity is ensured and validated because the record ID within a data record is compared with the record ID that is given as a parameter in the FARW when a record is accessed with a find or file macro. If the comparison fails, the access request is *not valid*.

Record ID Attribute Table (RIAT)

The record ID attribute table (RIAT) is a system table that holds the information about data organization that is necessary for data management by the TPF system. The RIAT is organized by and accessed by record ID. The RIAT contains information for both fixed records and pool records, such as:

- Logging characteristics

  Logging characteristics indicate whether or not a record is to be written to a real-time tape whenever a file type macro is processed for a record of the specified record ID. Logging is another way of ensuring database integrity by keeping alternate copies of important data records.
- Exception recording characteristics

  Exception recording characteristics indicate whether or not the record is to be written to a real-time tape if a file type macro has been processed after the record is captured by the TPF system utility file capture and restore (sometimes referred to as capture/restore), which is used to back up the online module (database).
- User exit characteristics

  User exit characteristics permit an application to dynamically modify the data record attributes that are assigned to a record ID at system generation time.
- VFA candidacy characteristics

  VFA candidacy characteristics specify whether a record is a VFA candidate. If the record is a VFA synchronization candidate, the data is also specified whether it is delay file or immediate file. (VFA is discussed in more detail in “Virtual File Access (VFA)” on page 138.)
- Module record caching candidacy characteristics

  Module record caching candidacy characteristics specify whether or not a record is a candidate for module record caching. If it is a candidate, the data specifies whether a file type macro request for a record of the specified record ID results in an I/O operation that is a fast write or retentive write. Record caching is discussed in more detail in “Retain Module Records in Module Cache Memory” on page 142.
- Lock maintenance characteristics.

  For a record that is a VFA candidate, lock maintenance characteristics specify whether the lock is held in the record hold table or the hold table associated with an external lock facility (XLF). See “Record Hold Table and XLF Lock Table” on page 144 for more information about XLFs.

A record ID must be specified for each pool record type (such as SSTx or LDPx) incorporated in a TPF system. This RIAT item is used to identify the appropriate pool record type when a get file storage macro request is invoked to obtain a pool record. In addition, the RIAT contains the following additional information about pool records:

- Size
- Record longevity
- Duplication
- Logical device type.

Record Addressing

Record addressing is the process of converting a symbolic file address, which is used by an application program, into its physical hardware address, which is used by the TPF system. To eliminate some processing overhead on each reference to a record, this conversion is done in stages. Various system services and tables are used in this conversion process.

Record Addressing Conversion Services (FACE, FACS, FACZC, and FAC8C)

There are several facilities available to convert symbolic addresses to physical addresses. They are:

- FACE program
- FACS program
- FACZC macro
- FAC8C macro.

The difference between FACE and FACS is that the record type input to FACE is specified as a numeric value whereas the record type input to FACS is specified symbolically. The FAC8C macro is similar to calling the file address compute program (FACE or FACS). The difference is that the FAC8C macro provides an interface to FACE for programs that use the IFAC8 parameter block. The IFAC8 parameter block only accepts an 8-byte file address as output. The FAC8C macro also accepts an 8-byte ordinal number as input.

The FACZC service routine operates like FACE, FACS, and FAC8C, except that it provides access to all fixed records no matter where the FACZC macro is called.

These facilities make use of the file address compute table (FCTB).

File Address Compute Table (FCTB)

The file address compute (FACE) table (FCTB), a system table, is a centralized source of information about file addresses. It provides the information necessary to do the following:

- Convert a record type and ordinal number into a file addressing scheme called file address reference format (FARF). This conversion applies to both fixed-file records and pool records.
- Convert a FARF address into its physical record address in module cylinder head record (MCHR) format.

For fixed addresses, the FACE table is accessed by the FACE and FACS programs and the FAC8C and FACZC macro service routines. For pool addresses, the FCTB is accessed by the Get File Storage macro service routines.

The FACE table is created by the offline FACE table generator program using various specifications defined by the user installation.

Application Record Addressing

The application procedure for referring to a fixed record is different from that for referring to a pool record.

- To refer to a *fixed record*, the application program specifies the fixed record type name and ordinal number of the record. In principle, the fixed record type name is converted into an FCTB index, which is used to locate the start of the fixed record area for the record type. The ordinal number is then used to locate the desired record within the fixed record type.
- Before referring to a *pool record*, the application program must request one by specifying a record ID associated with the pool area from which the record is dispensed. The TPF system returns the address of a pool record, which is used by the application program to reference the pool record.

Record Accessing

An application program's request to find or file a record is accomplished through the use of the find and file macros by specifying the symbolic file address of a record in a file address reference word (FARW) that is the application program interface (API) in the ECB. The TPF system's find and file macro service routines perform the additional conversion necessary to obtain the physical address by using symbolic file address, the FACE table (FCTB), and the module file status table (MFST).

The record ID provided as input to the macro request is used by the system to access the RIAT table to determine virtual file access (VFA), logging, and exception recording characteristics.

File Address Reference Format (FARF)

The file address reference format is the method used by the TPF system to symbolically address fixed and pool records. The formats of these symbolic file addresses have evolved over the years.

The 4-byte file address defined by FARF is a basic element of data organization in the TPF system. So, as technology increases the capacity of modules, this 4-byte file address must continue to represent all of the addressing capacity of modules. There are four file address reference formats: FARF3, FARF4, FARF5, and FARF6. These allow the TPF system the flexibility of adapting to module technology.

- File address reference format 3 (FARF3) is a format from the past and, because of its characteristics, is limited in its addressing capacity and flexibility.
- File address reference format 4 (FARF4) uses almost all of the addressing capacity allowed by a 4-byte symbolic file address. This format is primarily intended as a migration stage from FARF3 to FARF5.
- File address reference format 5 (FARF5) uses all of the addressing capacity in a 4-byte symbolic file address.
- File address reference format 6 (FARF6) is an 8-byte symbolic file address.

FARF4 and FARF5 are more flexible than FARF3, and therefore lead to less unusable addressing capacity than FARF3.

FARF6 is an 8-byte file address that is used to address 4-K long-term duplicated pools only. It provides for additional pool address expansion and can be used with FARF3, FARF4, and FARF5 addresses.

The amount of addressing capacity available is directly related to the number of dedicated control bits in each address format. FARF3 has the most control bits. Moreover, the same format is used for both fixed records and pool records.

FARF3, shown in Figure 29, provides for 2^{26} pool addresses per pool type and provides for 2^{28} fixed file addresses. FARF4, shown in Figure 30, supports a file address capacity of 2^{30}, and FARF5, shown in Figure 31 on page 108, supports a file address capacity of 2^{32}. Additionally, FARF4 and FARF5 addressing capacity can be spread between pools and fixed records whereas FARF3 dictates a certain amount of addressing capacity to pools and a certain amount to fixed records.

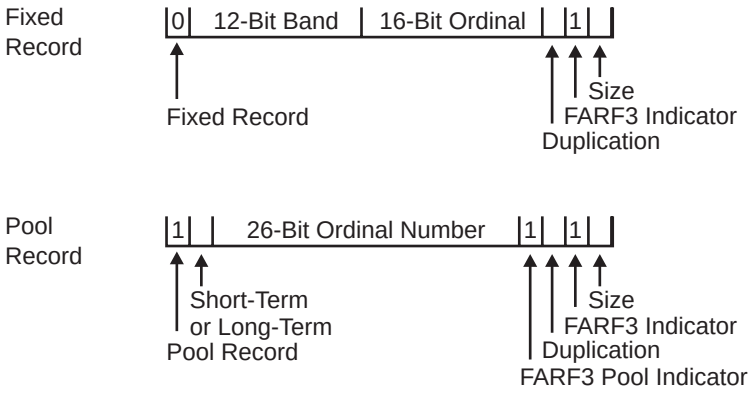


Figure 29. FARF3 Format

As shown in Figure 29,

- For a fixed record, a FARF3 address is made up of control bits, a band number, and an ordinal number.

  The band number is a unique random value between 0 and 4095 that is associated with a fixed record type.
- For a pool record, a FARF3 address is made up of control bits and an ordinal number.

The control bits are described previously. The ordinal number is an ordinal number within record type.

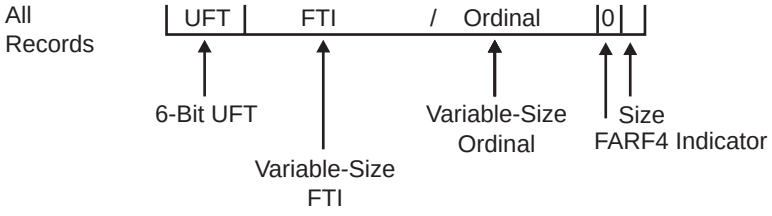


Figure 30. FARF4 Format

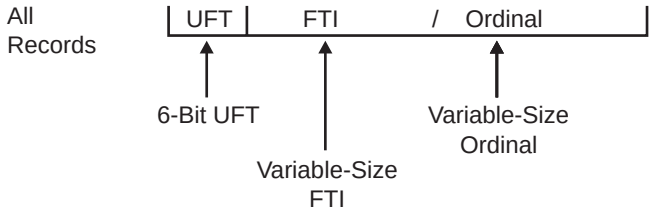


Figure 31. FARF5 Format

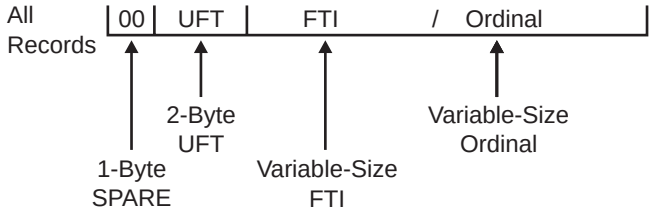


Figure 32. FARF6 Format

A FARF4 address, shown in Figure 30 on page 107, is made up of a universal format type (UFT), a format type indicator (FTI), an ordinal number, and control bits. The UFT and FTI are like the band number in FARF3. They are unique random values associated with a type of record (pool or fixed). The ordinal number is an ordinal number within a descriptor. A FARF5 address, shown in Figure 31, contains the same types of fields as a FARF4 address except there are no control bits.

Unlike the 32-bit FARF5 address, the FARF6 address shown in Figure 32 is 64 bits long. The FARF6 UFT is 2 bytes long versus the 6-bit UFT for FARF4 and FARF5 addresses. The FARF6 FTI and ordinal fields also have size restrictions that do not apply to FARF5 addresses. The FARF6 FTI must be at least 1 byte and can be no larger than 3 bytes. The ordinal field must be at least 2 bytes, but no larger than 4 bytes.

Only two address formats from the FARF3, FARF4, and FARF5 set can be generated in a TPF system at the same time: either FARF3 and FARF4, or FARF4 and FARF5. Moving between migration stages (from FARF3/FARF4 to FARF4/FARF5) in an online system requires that you load a new FACE table. FARF6 addresses are independent from the other FARF address formats and can be generated at any time.

Record Holding

The TPF record hold facility reserves a data record for the exclusive use of an Entry to make a modification to the record. Recall, from “Reentrant Programs” on page 39, the distinction between a program and a process; an Entry is a TPF process, the *animation* of a program. Several different Entries can be the *animation* of the same program. A record hold table consists of a list of records being held by active Entries in a CPC.

When an Entry issues a find and hold macro, the record hold table is checked to determine if the record is already held. If not, the file address of the requested record is placed in the record hold table and the request is serviced. However, if another Entry is holding the requested record, the current request is not serviced until the record is not held by the other Entry. The Entry requesting the record is

blocked and observes an I/O delay; however, multiprogramming is employed to keep an I-stream engine occupied with useful processing.

All record hold requests are serviced on a first-come, first-served basis. A file address is removed from the record hold table when an *unhold* request is processed and there are no further requests to hold this record. The fact that a record is being held does not prevent another Entry from reading the same record. The record hold table is checked only when servicing *find and hold*, *file and unhold*, and *unhold* macro requests.

In the TPF system, a data record is locked (held) at the record level. Applications ***should*** observe the rule of holding only one record at a time, thereby avoiding deadlock. (Observe that the TPF system minimizes overhead, in the name of performance, by ***not*** enforcing this rule.) Applications with a requirement to lock data represented by complex chains of records are not easily accommodated within the TPF system. The chances of an application program encountering an I/O delay because of a data record being locked can be minimized through application design, through the allocation of data records across all modules, and by locking on single records rather than on larger *data sets*. (Contrast this with the IBM MVS system that *locks* on a data set through the OLD parameter in the job control language (JCL) that references the data set.) Practice has shown that although record hold checking is performed on each hold-type request, seldom does a held record cause the processing of another Entry to be delayed.

Record holding within an environment of a single CPC is conceptually simple and permits Entries running in a multiprogramming environment to share the same database. Perhaps a better viewpoint within the TPF system is to consider the programs that process related but essentially independent data. An analogy is a large dining room, a large buffet table full of multiple selections of a variety of food (the *data*), plenty of tables to seat hungry people (the *programs*), where everyone gets plenty to eat with few queues, little waiting, and vanishing hunger.

A review of the material in “Central Processing Complex (CPC)” on page 37 is recommended at this point. In particular, review Figure 13 on page 45.

Within a loosely coupled complex, record holding must be coordinated among the CPCs that form the complex. Conceptually, the idea is the same as coordination among I-stream engines within a CPC and among Entries in an I-stream engine. That is, a large database is accessed by an identical set of application programs available for execution (*animation*) in each of several CPCs. However, a private main storage in each of the CPCs adds some complexity to the underlying details. Simply placing file addresses, essentially lock indicators, in a main storage record hold table is no longer adequate to communicate the need for exclusive control of a record. Because the records are still located on shared modules and any indicator held in main storage is private to a CPC, all Entries in the complex must have access to the lock indicator, but not every Entry is processed by the same CPC. Therefore, the main storage of a CPC cannot be used as the communication vehicle among these Entries. The external lock facility (XLF) and associated software support (introduced in TPF System Processing Milieu) moves the record hold table to storage in a shared external facility that is accessible by all CPCs. The communications vehicle is now represented by shared storage in this external facility.

Module File Status Table

The module file status table (MFST) is a system table used to identify each module to the TPF system. For each module, the MFST contains status information (whether the module is accessible or not), its logical device type, and the linkage to its hardware address for use by the following functions:

- Validation of FIND/FILE requests
- Queuing and I/O operations
- Module hardware maintenance
- Error recovery
- Data collection.

The MFST is a device dependent table, which means it contains information about each of the four possible logical device types (DEVA, DEVB, DEVC, and DEVD) generated in a TPF system.

Once the online modules have been initialized (formatted) for online use, the module's volume serial number is used as a module identifier that maps to a symbolic module number. This symbolic module number is an index into the module file status table (MFST).

Another field in the MFST is the symbolic device address (SDA), which is related to a *physical* or *hardware* address for a module. The assignment of an SDA to a symbolic module number in the MFST is always done during a system restart or during an initial program load (IPL) with a procedure called roll call. Figure 33 on page 111 shows the relationship between SDAs and symbolic module numbers.

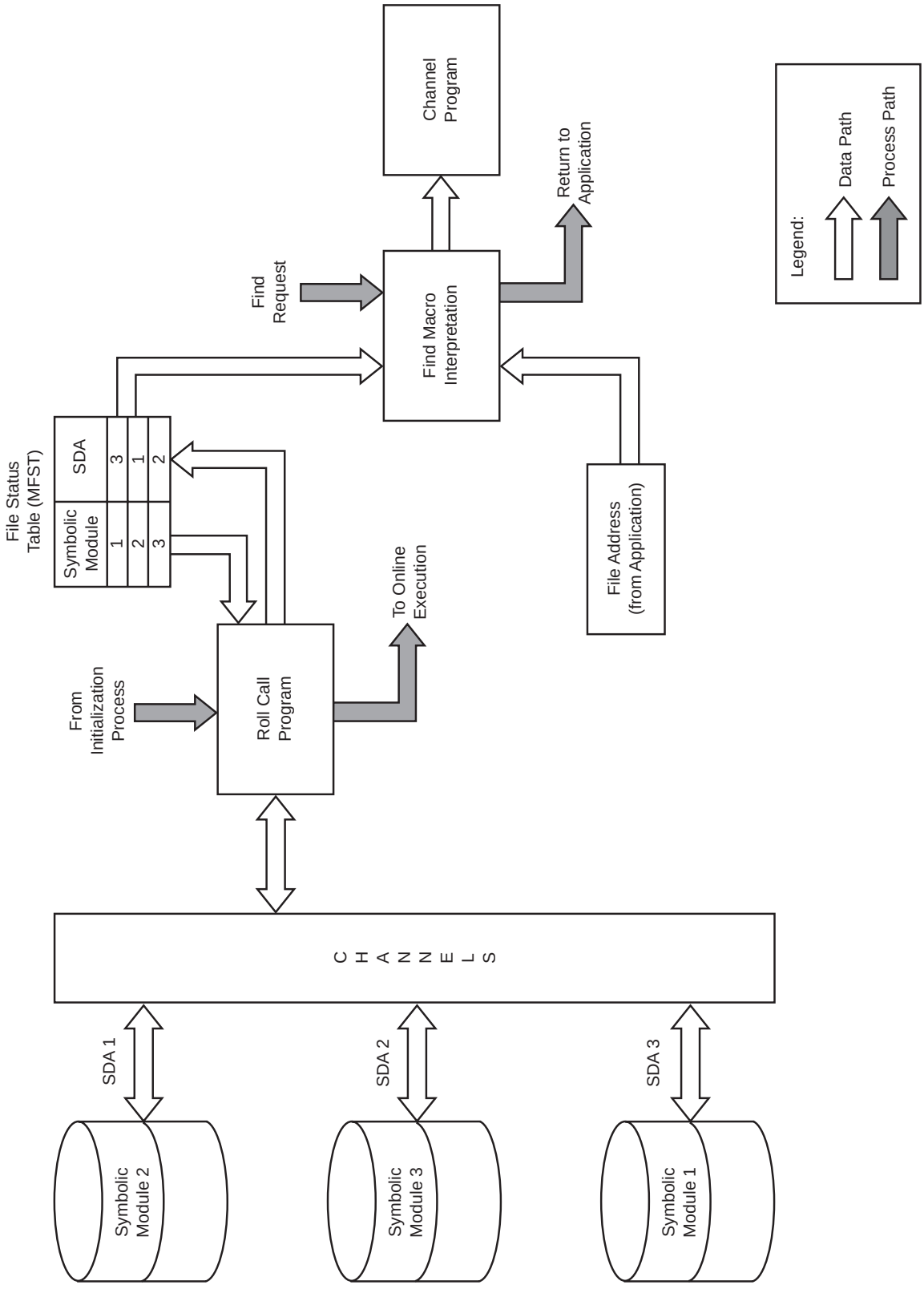


Figure 33. Symbolic Module Numbers

Record Allocation

The purpose of module storage is to hold programs and data that are requested by Entries and then transferred to main storage for use. The performance of a TPF system is primarily dependent on the number of file storage requests and the time required to seek and transfer them from a module to main storage. The number of requests is largely determined by application design. The seek and transfer time includes the amount of time taken to process the request by the TPF system, the time in queue for the device, the time required to find the requested data on the physical device, and the time to transfer the data to main storage. Because a device can only handle one request at a time, multiple requests for a particular device must be queued.

The seek and transfer time are based on device characteristics. The queuing time, however, is dependent on data organization. A first step in reducing the queue for any device can be to design a system that approaches equal queue sizes for each device. The average length of the queue on a device can then be reduced by increasing the number of devices.

The factors governing database organization design are:

- The capacity of the module in relation to the quantity of data to be contained
- The ability of the module hardware and the TPF system to handle requests at a rate consistent with the objectives of system performance.

As queuing increases, the data-request time increases, the life of an Entry increases, and a greater demand is made on working storage. This results in either reduced system performance as viewed by an end user or a system failure because of lack of system resources (working storage).

The organization of the TPF system's data is intended to distribute the records associated with a set of data, called a record type, over physical file storage to reduce queuing time at the module. Therefore, when Entries access logically adjacent records within a set of data, each record is obtained from a different physical module device.

The fixed record types used by the applications that run in the TPF system are defined during system generation. Contrast this with the batch orientation of data definitions in the IBM MVS system where data sets are defined with a combination of JCL, source code, and supervisory service routines; that is, the complete description of the data set is always deferred until job execution. In the TPF system, the definition of all fixed file records at system generation has the advantage of eliminating the overhead of describing the characteristics of a set of data each time a unit of work is processed. This forces all the files (that is, sets of data) that are required by the online applications to be physically in place and always available, which is simply a characteristic of an online system that runs in real time and accepts random input.

The FACE table (FCTB) is the system table that identifies, for both pool and fixed records, where the various record types are allocated on the physical module devices. The FACE table points to the origins of the various record types, called base addresses.

The functions of file allocation and file address conversion occur at distinctly different times. The allocation occurs once, when a FACE table is generated, to produce the base addresses of the various record types. The file address

conversion to a physical file address occurs each time an application (Entry) issues a find or file macro request for a data record. This conversion process uses the FCTB generated by the allocation function.

Before proceeding, you should review the concept of *horizontal allocation* shown in Figure 26 on page 98. The purpose of this method of record allocation is to allow a larger number of concurrent accesses to any particular record type, thereby reducing the chance of excessive queuing.

Each logical device type (such as DEVA and DEVB) has its own organization based on the physical characteristics of each device. The records within a given record type are allocated on each logical device type according to the rules of that device type. The records within a single record type can be split across logical device types, allowing each portion to be placed on a different type of device (see Figure 34).

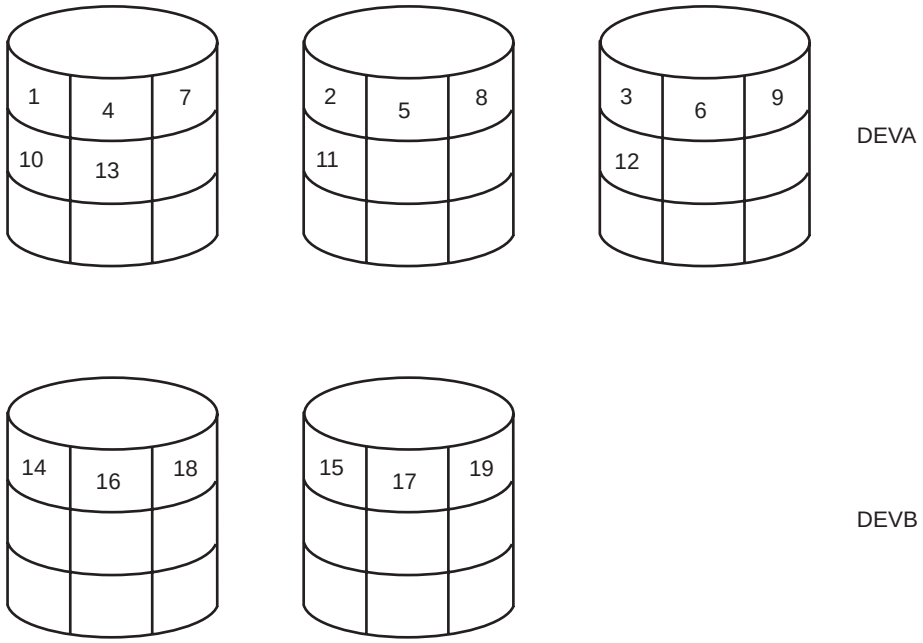


Figure 34. Record Allocation Across Different Types of Module Devices

The TPF system views the entire file space as a repository for holding 4K, 1055-byte and 381-byte records. Within a given record type (for example, 4K fixed records), the TPF system is capable of naming, through the use of database ordinal numbers (DBON), some maximum number of records.

Consider the representation of the fixed file space of n records (database ordinal numbers 0 through n-1) in Figure 35. The number of record types is given as m, where m is less than n.

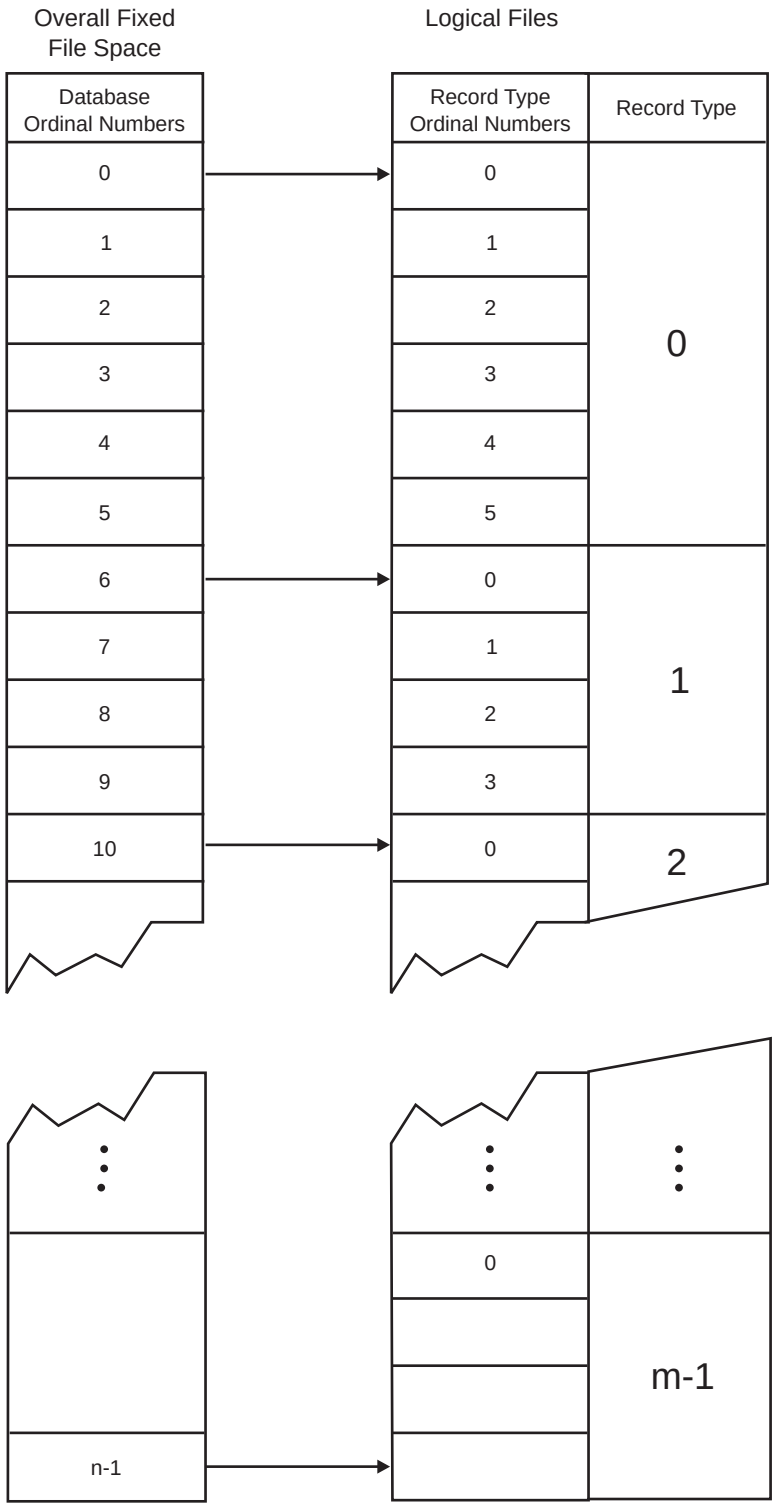


Figure 35. Total Record Space. Assuming Only One Size Record.

To allocate fixed file space means dividing the entire fixed file space into subgroups of record types, where a subgroup consists of either all small, all large, or all 4K fixed records. The allocation is accomplished by creating a FACE table that associates the base address, called a beginning database ordinal number (DBON), with each record type. Each DBON, after the first one, accounts for the number of

records assigned in the preceding record types. For example, in Figure 35 on page 114 record type 1 begins at DBON 6 and four records are allocated to record type 1.

The next record type (record type 2) begins at DBON 10. In general, each record type is associated with some unique DBON. Notice that the second record (record type ordinal number 1) of record type 1 is at DBON 7 of the overall fixed file space. It is important to notice that:

- A fixed file area for holding record types consisting of records of the same size has a set of database ordinal numbers (0 through n-1).
- Each fixed record type has a set of DBONs equal to the number of records designated for that record type.

There is a similar procedure for allocating the pool file space. The record type for a pool file is based on the pool record type (such as SSTx and LDPx).

The online physical file storage is allocated for the different record sizes in proportions that are primarily determined by application design. The sum of allocations of all pool and fixed records comprises the entire online physical file space.

The TPF system ensures that the sequential database ordinal numbers (DBON) do not map into physically adjacent records. It is assumed, in general, that the horizontal allocation of records of a given record type throughout the file storage permits a greater chance for simultaneous accessing than the more conventional, vertical allocation, as shown in Figure 26 on page 98. This strategy of allocating records of the same type throughout physical storage is based upon the assumption that many Entries are accessing a shared database. Therefore, the TPF system ensures at least a chance for simultaneous accessing of records within a record type by forcing the records within a record type to be spread across modules. The way this is accomplished is simple in concept but somewhat complicated in detail, and is beyond the scope of this publication.

Relationship Between DBON and Physical Address

The allocation of records associates successive database ordinal numbers (DBON) with successive modules, cylinders, heads, and records on modules. Consider the example represented by Figure 36 on page 117 that shows a configuration of two modules, two cylinders, two heads per module, and five records per track. The numbers in parentheses represent database ordinal numbers (DBON). The numbers without parentheses represent the device addresses (that is, the names of physical locations). Consider all access arms positioned to the same cylinder on all modules. Then, an allocation is stated as follows:

- Pick head 0 and record 0 and assign the record addresses module by module, and when the last module (module 1, in this case) is used, start at the next record and once again assign records module by module.
- When a track (that holds 5 records, in this case) is used (fully allocated), start at the next head (head 1, in this case), record 0, module 0, and continue.
- When all heads (only two, in this case) are used, start at the next cylinder, head 0, record 0, module 0, and continue.
- When all cylinders (only two in this case) are used, the disk is fully allocated.

This allocation defines the rules that the TPF system employs to map a data record reference (file address) into an address used by hardware. A record type can be allocated across different logical device types with physical, device-dependent

values (that is, the number of modules, cylinders, heads, records). The physical, device-dependent values are held in system tables (such as the MFST, FCTB, and pool directories) utilized by programs that perform address mappings.

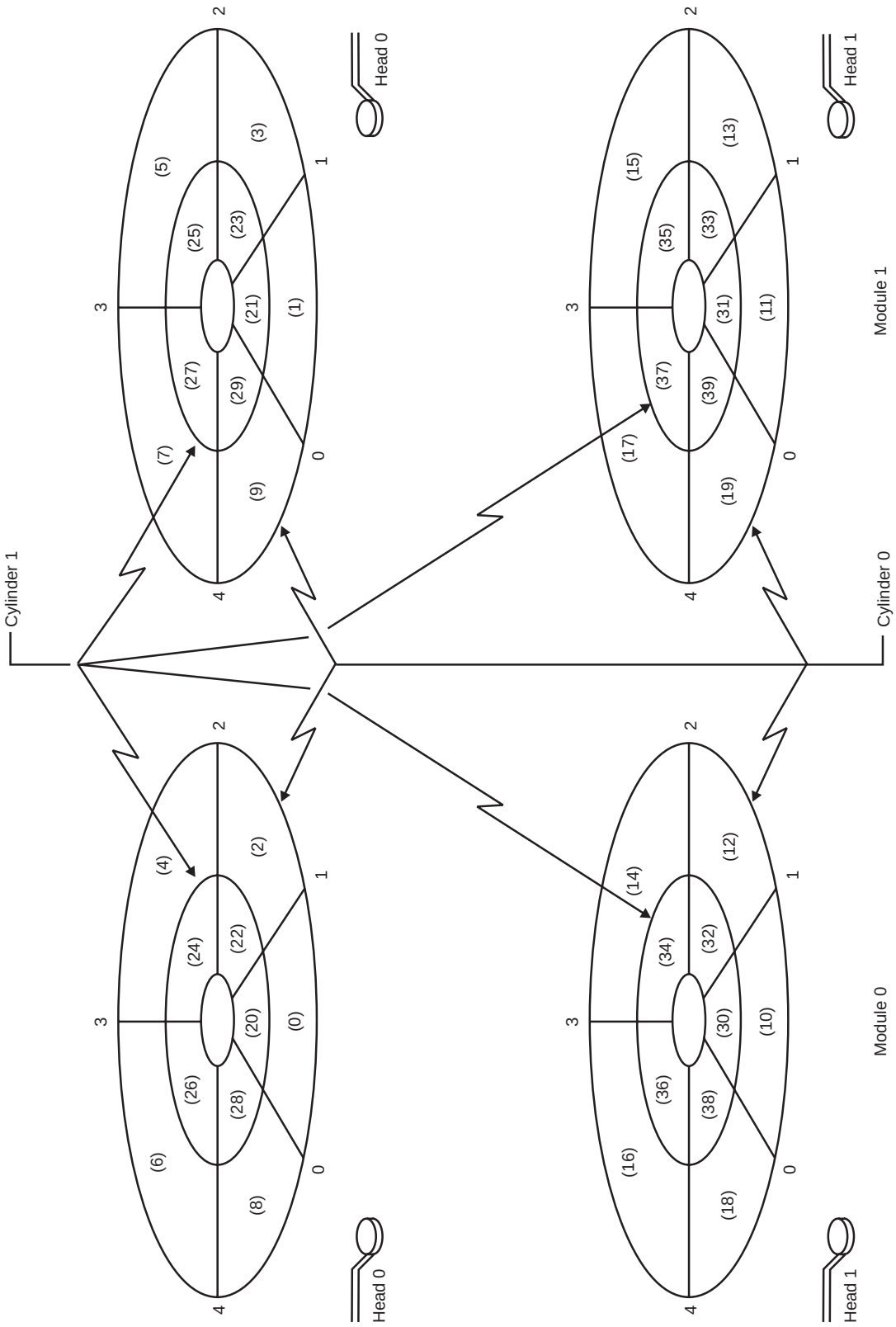


Figure 36. Allocation Example

Record Mapping

The combinations of find and file macro service routines, the file address conversion facilities (FACE, FACS, FAC8C, and FACZC), and the get file storage macro service routines must perform the inverse of the allocation process; that is, to convert a record type and an ordinal number into an address used by hardware.

To compute a file address of a data record means the conversion of a record type and an ordinal number into a physical file location. This is accomplished by:

- A table lookup to find the record type within the FACE table
- The lookup using the record ordinal number to find the FARF address of the record
- The conversion of the FARF address into a DBON within a fixed file space or pool file space
- The conversion of the DBON into a physical file address consisting of module, cylinder, head, and record.

To determine the physical address for record type 1, ordinal number 3, the TPF system must:

- Obtain the base DBON for record type 1, which is DBON 6 (see Figure 35 on page 114).
- Add the given ordinal number (which is 3) to the base DBON, which results in DBON 9.
- Convert DBON 9 to a physical address using Figure 36 on page 117 to conclude that M,C,H,R = 1,0,0,4.

The TPF system calculates MCHR by manipulating the DBON of the desired record as follows:

```
DBON/H*R*M    =    Q1, E1    (Q1 = Cylinder number)
   E1/R*M     =    Q2, E2    (Q2 = Head number)
   E2/M       =    Q3, E3    (Q3 = Record number, E3 = Module number)
```

Where:

Term	Description
Q	Quotient.
H	Number of heads per cylinders associated with the type of device.
R	Number of records per head associated with the type of device.
M	Number of modules per database associated with the type of device.
E	Remainder.
DBON	Database ordinal number of the data record in the database.

To use the example just described:

```
Where:  DBON = 9  DBON/H*R*M  =  9/20  =  0,9    C = 0
                     E1/R*M   =  9/10  =  0,9    H = 0
                     E2/M     =   9/2  =  4,1    R = 4
                                                 M = 1
```

Therefore, MCHR = 1, 0, 0, 4 (as shown by Figure 36 on page 117).

C, H, R, and M are constant values that depend on the geometry and the number of modules. Table 5 on page 119 uses the following sample values: C=2, H=2, R=5,

and M=2. *c*, *h*, *r*, and *m* are variable values that depend on the value of the DBON and have maximum values of C-1, H-1, R-4, and M-1 respectively.

Where:

Term	Description
c	Cylinder number in the module.
h	Head number in the cylinder.
r	Record number in the head.
m	Module number in the database.

Table 5. Determining the Algorithm Relating the Cylinder, Head, Record, and Module to the DBON

Record Number (DBON)	Cylinder (c)	Head (h)	Record (r)	Module (m)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	0	2	0
5	0	0	2	1
6	0	0	3	0
7	0	0	3	1
8	0	0	4	0
9	0	0	4	1
10	0	1	0	0
11	0	1	0	1
12	0	1	1	0
13	0	1	1	1
14	0	1	2	0
15	0	1	2	1
16	0	1	3	0
17	0	1	3	1
18	0	1	4	0
19	0	1	4	1
20	1	0	0	0
21	1	0	0	1
22	1	0	1	0
23	1	0	1	1
24	1	0	2	0
25	1	0	2	1
26	1	0	3	0
27	1	0	3	1
28	1	0	4	0

Table 5. Determining the Algorithm Relating the Cylinder, Head, Record, and Module to the DBON (continued)

Record Number (DBON)	Cylinder (c)	Head (h)	Record (r)	Module (m)
29	1	0	4	1
30	1	1	0	0
31	1	1	0	1
32	1	1	1	0
33	1	1	1	1
34	1	1	2	0
35	1	1	2	1
36	1	1	3	0
37	1	1	3	1
38	1	1	4	0
39	1	1	4	1

The following algorithm maps the values DBON, *c*, *h*, *r*, *m*, C, H, R, and M for Table 5 on page 119:

```
DBON = c(H*R*M) + h(R*M) + r(M) + m
```

Example 1

What is the value of the DBON if:

```
c = 1
h = 0
r = 2
m = 1
```

```
DBON = c(H*R*M) + h(R*M) + r(M) + m
     = 1(2*5*2) + 0(5*2) + 2(2) + 1
     = 20 + 0 + 4 + 1
     = 25
```

Example 2

Determine the physical file address values for *c*, *h*, *r* and *m* if the DBON = 24.

For the following equation:

```
DBON = c(H*R*M) + h(R*M) + r(M) + m
```

dividing each side of the equation by (H*R*M) provides the following:

```
24/20 = c + (h(R*M) + r(M) + m)/20 = c, E1
c is the quotient, E1 is the remainder.
E1 = h(R*M) + r(M) + m not (h(R*M) + r(M) + m)/20
c = 1, E1 = 4
```

For the following equation:

```
E1 = h(R*M) + r(M) + m
```

dividing each side of the equation by (R*M) provides the following:

```
4/10 = h +  (r(M) + m)/10 = c, E2
h is the quotient, E2 is the remainder.
E2 = r(M) + m
h = 0, E2 = 4
```

For the following equation:

```
E2 = r(M) + m
```

dividing each side of the equation by M provides the following:

```
4/2 = r + m
r is the quotient, E3 is the remainder which equals m.
E3 = m
r = 2, E3 = 0, m = 0
```

Therefore, when the DBON = 24, the physical file address values *c*, *h*, *r* and *m* are as follows:

Table 6. Physical File Address Values When DBON=24

Record Number (DBON)	Cylinder (c)	Head (h)	Record (r)	Module (m)
24	1	0	2	0

Duplication of Records

Record duplication addresses the design objectives of the TPF system of:
- Performance
- Database integrity

by providing the ability to maintain two copies of critical data records.

Remember from “Record Duplication” on page 102 that there are two copies of a record on the database, which are referred to as the primary record and the backup record (sometimes called the duplicate record or dupe).

There is a trade-off between performance and resources (modules hardware) versus system reliability and integrity when dealing with record duplication because there is some overhead to manage two copies of a data record (for example, two I/O commands are issued). However, this is offset with an improvement in accessing the data record because there is an alternate path to the data in case there is disk contention.

Primary and backup records must be allocated to the same logical device type. Allocation of fixed and pool data can be specified as:
- Non-duplicated — None of the records are duplicated on the logical device type
- Selectively duplicated — Selected (critical) records are duplicated on the logical device type
- Fully duplicated — All records (fixed and pool) are duplicated on all of the logical device types generated in the user installation.

An entire database can be duplicated, in which case the attribute of duplicate is redundant. Only long-term pool records can be duplicated in a selectively duplicated module configuration, which is the significance of SDPx, LDPx, and 4DPx.

The allocation procedure assigns two database ordinal numbers (DBONs) to a duplicated record, each to different modules.

In a selectively duplicated system, non-duplicated pool records are spread across all modules (primary and duplicated modules) while the duplicated pool records are allocated so that the primary pool records are located on the primary modules and the backup pool records are located on the duplicate modules.

As a matter of fact, in a selectively duplicated system for a fixed file that is not duplicated, backup records are reserved on the duplicate module in order to preserve the mapping for the remainder of the module. For such records, only a single write (for the primary record) is performed by the file macros.

Non-duplicated data and selectively duplicated data are supported to remain compatible with the past. Fully duplicated is always recommended.

Records are always allocated on a module to module basis, which means the backup record within a record type (either pool or fixed) is assigned to the same relative position as the primary record on an alternate disk module. Duplication of records can only be utilized with configurations that have an even number of modules because the primary and backup copies of records must appear at the same location on the primary and duplicate modules respectively.

Figure 37 shows the way of allocating duplicate records across the modules. The figure demonstrates the relationship of the primary and backup records within a record type; the figure does not imply that all primary or backup records of a given record type are allocated to the same module. The notations *Pn* and *Dn* are used to denote primary addresses and the corresponding backup (duplicate) addresses within a record type.

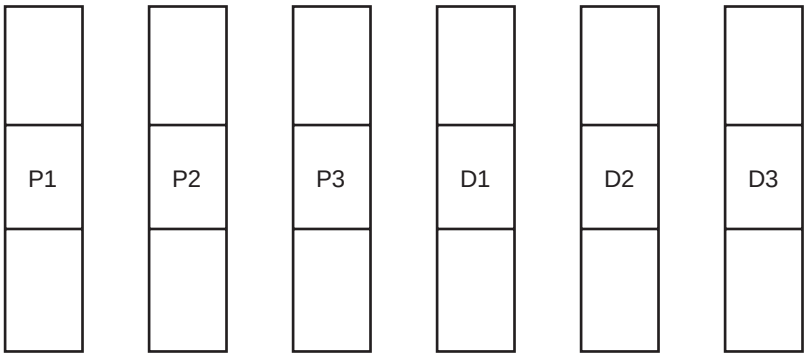


Figure 37. Module to Module Duplication

The file macro service routine updates both the primary and backup records if both exist. The find macro service routine retrieves either copy based on which disk device has the smallest I/O queue or the inability to retrieve one of the copies. Macros are provided for application programs to selectively read or write only the primary or duplicate copy of a record.

In a loosely coupled environment, only the prime record is retrieved because the *record hold table* for the LC complex is contained in the external lock facility (XLF).

Pool Directories

Pool file storage is managed through pool directories. There is a pool directory for each of the different types of pools as determined by the following attributes:

- Longevity (intended length of time in use)
- Record size

- Duplication.

An example of a pool type is small short-term (SSTx) (see “Pool Record Types” on page 103). The same pool type can exist on different logical device types. Separate pool directories exist for each pool type on each logical device type.

For efficiency purposes, within a pool directory, the TPF system maintains a single bit for each pool record that indicates the availability status of the associated pool record. The relative position of the bit within a pool directory determines the ordinal number within a pool type of that record, as shown in Figure 38 on page 124. This combination of pool type and ordinal number is a pool record reference.

When file storage is requested, the get file storage macro service routine scans the directory of a relevant pool type for an available pool record (indicated by a 1). When matched, the relative position number in the directory is used as an ordinal number and the pool record is marked as dispensed (meaning *not available*) by setting the bit indicator to zero. This act is known as dispensing a pool record. The pool record reference to the available pool record is returned to the application that issued the get file storage macro request in a file address reference word (FARW) of its ECB.

This pool record reference is converted to a physical address by the find and file macro service routines.

The return file pool address (RELFC) macro service routine reverses the availability bit of a pool record in a pool directory. In the case of a release, (return) the pool record reference is converted into the relative bit position in the pool directory.

A pool directory is usually so large that it requires multiple data records to contain all of the pool record bits. These records cannot all be held in main storage during online execution. This requires a cycling of directories from file storage to main storage and back to file storage. This mechanism is described in more detail in “Directory Reordering” on page 129.

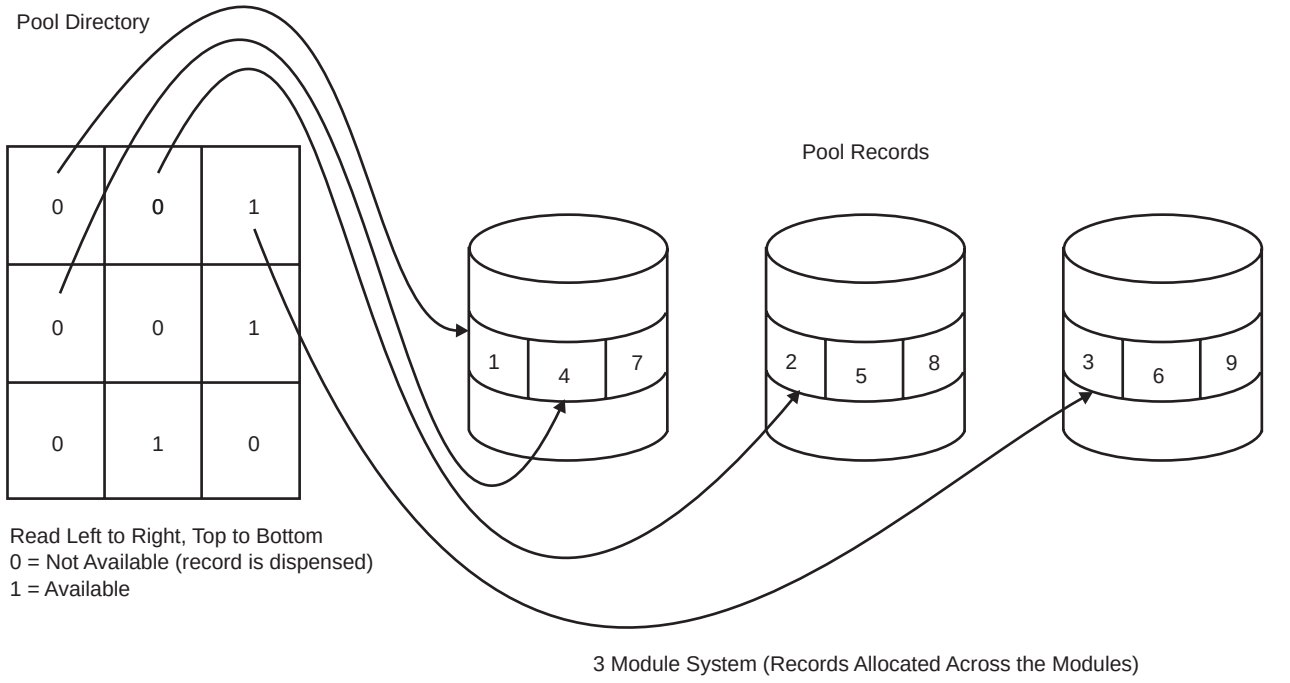


Figure 38. Example of a Pool Directory

Pool Management

The *life* of a pool record from the application point of view can be described as follows:

- A pool record address is obtained by issuing a get file storage macro request.
- The application program then uses the pool record to store data.
- A pool record is returned to the system by issuing a return file pool address macro request.

The pool macro service routines manage pool file storage by manipulating the pool record bits in the pool directories.

To dispense a pool record means to generate an ordinal number and mark the record as unavailable within a pool directory. The basic principle of dispensing pool records is straightforward. The detailed algorithms found in the system programs are slightly more complicated because of the following:

- Multiple pool record types are permitted.
- Pools can be split across different types of devices.
- A pool record type can be in noncontiguous file storage within a device type.
- A given pool record type can require several directory records.

An explicit description of the pool directory procedures is not given in this publication. However, the TPF system structures associated with these details are necessary to understand pool file management techniques.

Pool Section

See Figure 39 on page 125. The space allocated on a logical device type for a pool record type is called a pool section. Space for a pool record type can be allocated

across several logical device types, in which case there are several pool sections for the pool record type. (Recall the pool record type discussion in “Pool Record Types” on page 103.)

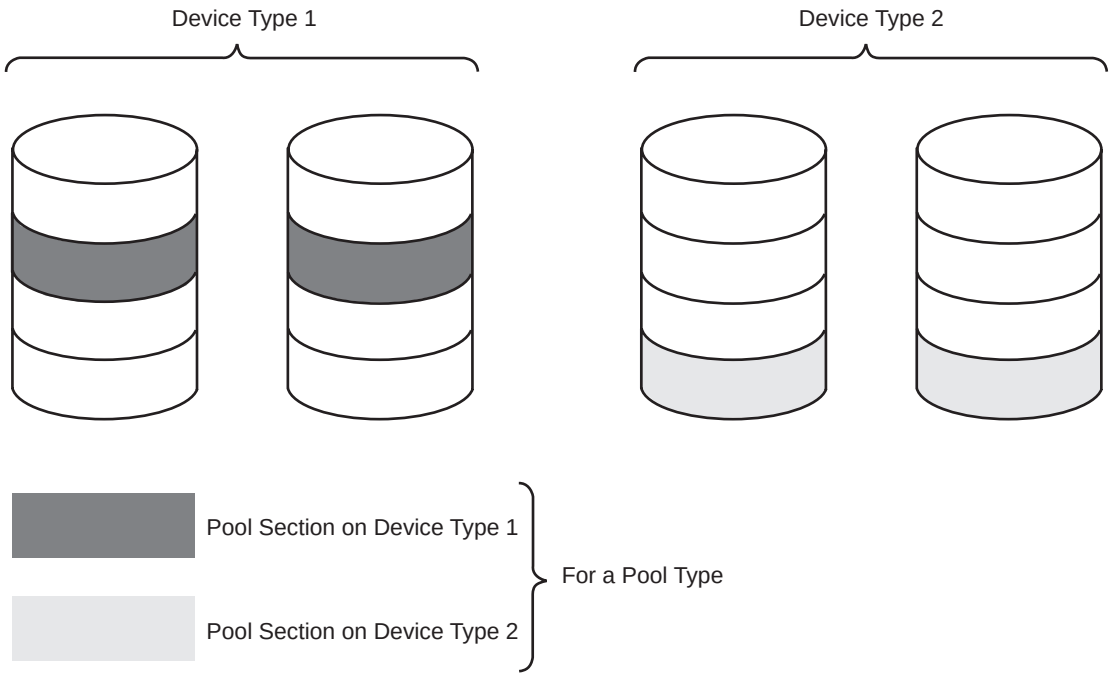


Figure 39. Pool Section. For a Pool Type.

Pool Segment

See Figure 40 on page 126. For a pool record type, several noncontiguous areas can be allocated per device within a logical device type. Each area within a pool section is called a pool segment; therefore, a section can be segmented.

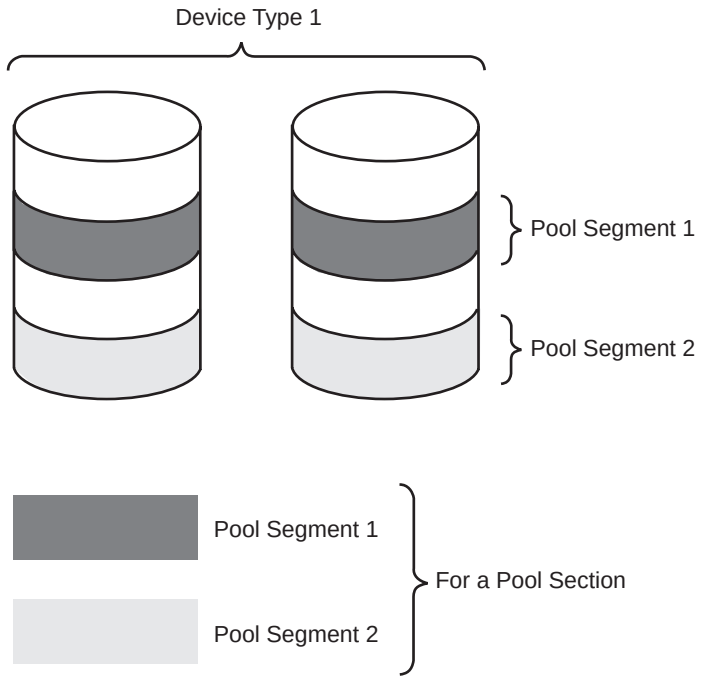


Figure 40. Pool Segments. For a Pool Section within a Pool Type.

Pool Directory

See Figure 41 on page 127. Recall that a pool directory contains the pool record availability bits for a pool record type. Each pool section of a pool record type requires at least one directory record. Depending on the number of pool records in a pool section of a pool record type, several directory records can be required. Each pool segment requires the start of a new directory record; therefore, a segmented pool section requires at least two directory records.

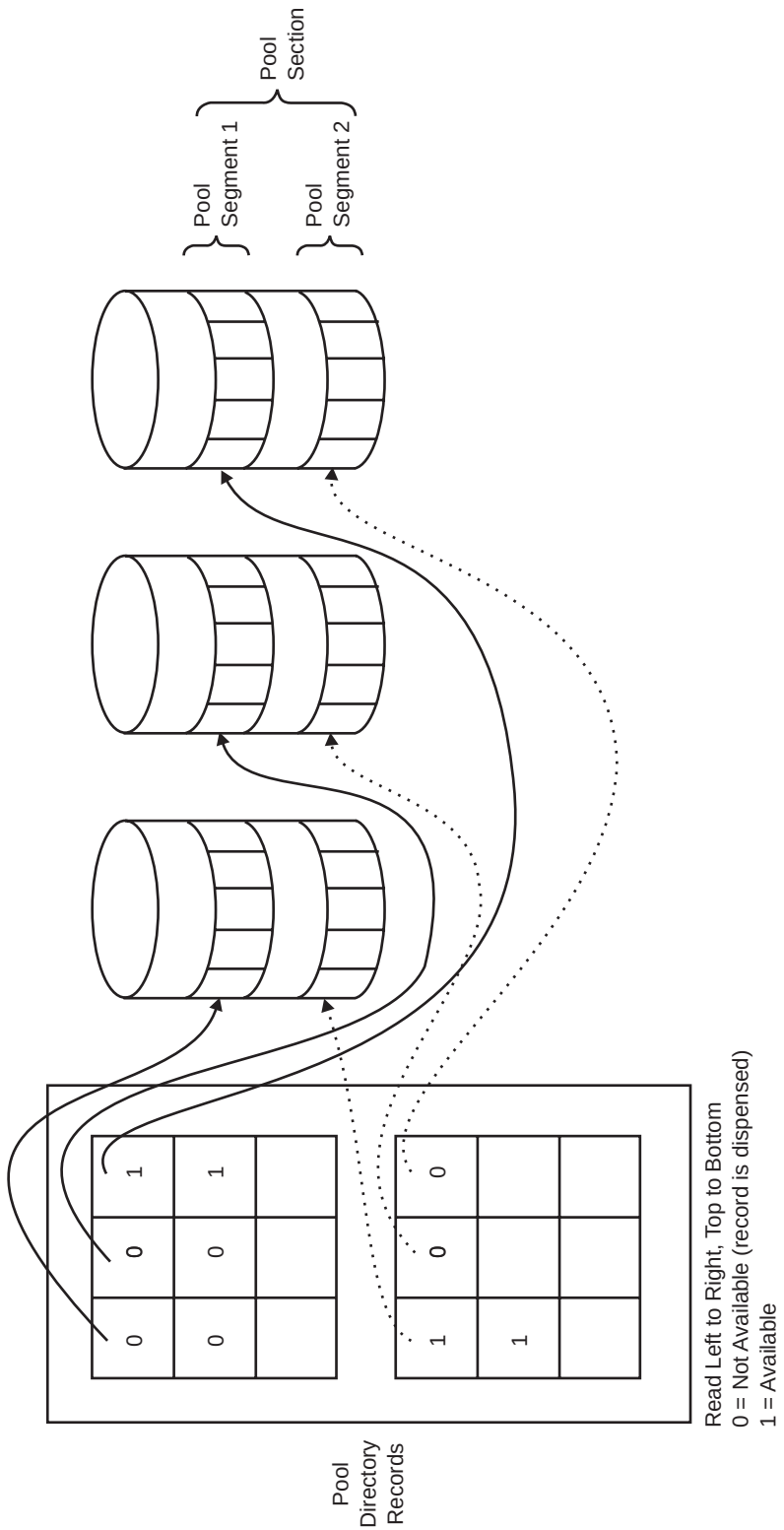


Figure 41. Pool Directory. For a Pool Type with Two Pool Segments.

Get File Storage

Application programs use a get file storage macro to request (obtain) a pool record in any one of the standard TPF record sizes. This dispensing of a pool record can be influenced by the pool management techniques of ratio dispensing or pool fallback.

Release File Storage

Application programs use the return file pool address request macro request to release pool records (that is, return pool records to the TPF system). At this point, the application program has fulfilled its obligation to return any pool file storage to the TPF system that it no longer requires.

The return file pool address macro service routine is responsible for resetting the appropriate bit within a pool directory record to indicate availability. For short-term pool records, the bit is reset immediately, and the pool record is immediately available for reuse. For long-term pool records, the setting of the bit is deferred because of a negative performance impact to the TPF system.

Because the rate of change to long-term pool directory records is much less frequent than that of short-term pool directory records, and the likelihood of any given long-term pool directory record being in main storage is low, processing of the return of a long-term pool record can be deferred until a period of low system activity. The negative performance impact can be explained by the overhead that would be incurred if it was necessary for the TPF system to retrieve a long-term pool directory record each time, or nearly each time, a long-term pool address was returned.

The processing done by the return file pool address macro service routine is based upon the longevity attribute as follows:

- When long-term pool records are released, the pool record references are written to released pool address (FC33) records.
- When short-term pool records are released, the pool record references are processed immediately. If the directory of the released reference is in main storage and is being used by get file storage processing for address dispensing, the appropriate availability bit is set to 1. Otherwise, the request to release a pool record is essentially ignored because all the record references of the appropriate directory record are eventually *released* when the directory is recycled (that is, all bits are set to one when the pool recycle time interval has elapsed; this is referred to as short-term pool recycling).

Ratio Dispensing

When the pool space consists of several types of devices, ratio dispensing is an attempt to spread the *in use* pool records across the types of devices to prevent I/O bottlenecks.

Therefore, by definition, for a particular pool type, there is a pool section for each logical device type. Ratio dispensing is the technique used to dispense addresses from the various pool sections of a pool record type based on a ratio factor. The ratio factor specifies the number of addresses to dispense from a pool section before selecting another pool section from which to dispense addresses.

Pool Fallback

If a depleted pool section (that is, a pool section containing no available addresses) is selected for address dispensing, an alternate compatible pool section is used if possible. This is called pool fallback. Selection of alternate pool sections for fallback processing is done based on a predefined schedule. Pool fallback schedules can provide short-term to short-term, short-term to long-term, or long-term to long-term fallback capability for depleted pool sections.

Directory Reordering

Multiple directory records are normally associated with each pool section. Consequently, a function called directory reordering can be invoked by the TPF system to process a request for pool file storage.

Directory reordering generally consists of scheduling retrieval (from file storage) of new directory records before the in-use directory records are depleted. This is done when the remaining number of available addresses in the in-use pool directory reaches a predefined critical level called the *reorder level*. At this point, an Entry is created that invokes the directory reorder mechanism to retrieve new directory records from file storage.

Short-Term Pool Recycling

By design, the pool sections for short-term pool records are recycled. This means if the last directory of such a pool section is depleted and if the time interval for recycling has elapsed, the section's first directory is again set up for dispensing with all bits in available status. See Figure 41 on page 127.

Therefore, applications should not utilize a short-term pool record for a longer period of time than the time interval between recyclings. This criteria is generally satisfied if the record is returned by the end of a TPF system transaction. Otherwise, pool record references could be dispensed a second time while still being used for a previous application request. The pool recycle time interval is dependent upon the application environment's use of short-term pool records and the amount of module space allocated to short-term pools.

Pseudo Modules

The TPF system provides the capability to allocate pool records to more modules than actually exist in the physical configuration. This capability facilitates pool space allocation when the database is expanded by adding more modules.

The addresses on these (future) modules, called pseudo modules, are marked as unavailable in the pool directories. When physical modules are added to the system, and because the pool availability bits are already allocated, it is a relatively simple procedure to begin to use the additional pool records. The procedure is performed by pool directory generation programs that mark the addresses as available.

Multiple Database Function (MDBF)

The multiple database function (MDBF) of the High Performance Option (HPO) feature permits the physical and logical separation of sets of data within the TPF online modules. An example is an airline's reservation application that supports several airlines, each with their own unique reservation records but which share hardware and system resources.

When sets of data are separated physically, a set of data is accessible by a subsystem. When sets of data are separated logically, a set of data is accessible by a subsystem user. When sets of data are separated physically and logically, a set of data is accessible by a subsystem user within a subsystem. Refer again to Figure 27 on page 99.

A subsystem can support multiple subsystem users. However, it is not necessary to define subsystem users.

The basic subsystem (BSS) is fundamental; at a minimum, the basic subsystem contains the sets of data required by the TPF system for its own operation and, therefore, must always exist. (A TPF system without MDBF is actually operating as a basic subsystem.)

Generating a TPF system with the multiple database function (MDBF) impacts the TPF system services and structures dealing with file management and message routing.

File Address Compute Table (FCTB)

There is one FCTB associated with each subsystem and its associated subsystem users. In addition, it contains a list of the subsystem user names for the subsystem. Each subsystem is given a subsystem ID (SSID) and each subsystem user within the subsystem receives a subsystem user ID (SSU ID). These IDs are used by the TPF system to control the access to data on the physical devices.

Furthermore, pool records are shared among all subsystem users on any given subsystem (in other words, stating this: each FACE table represents one shared allocation of pool records for a subsystem and all fixed record allocations for the subsystem users within the subsystem).

The FACE and FACS programs and the FAC8C macro return the file address for any fixed file record that is accessible from the subsystem user (SSU), processor, and I-stream engine that requested record addressing conversion services (see Figure 42 on page 131).

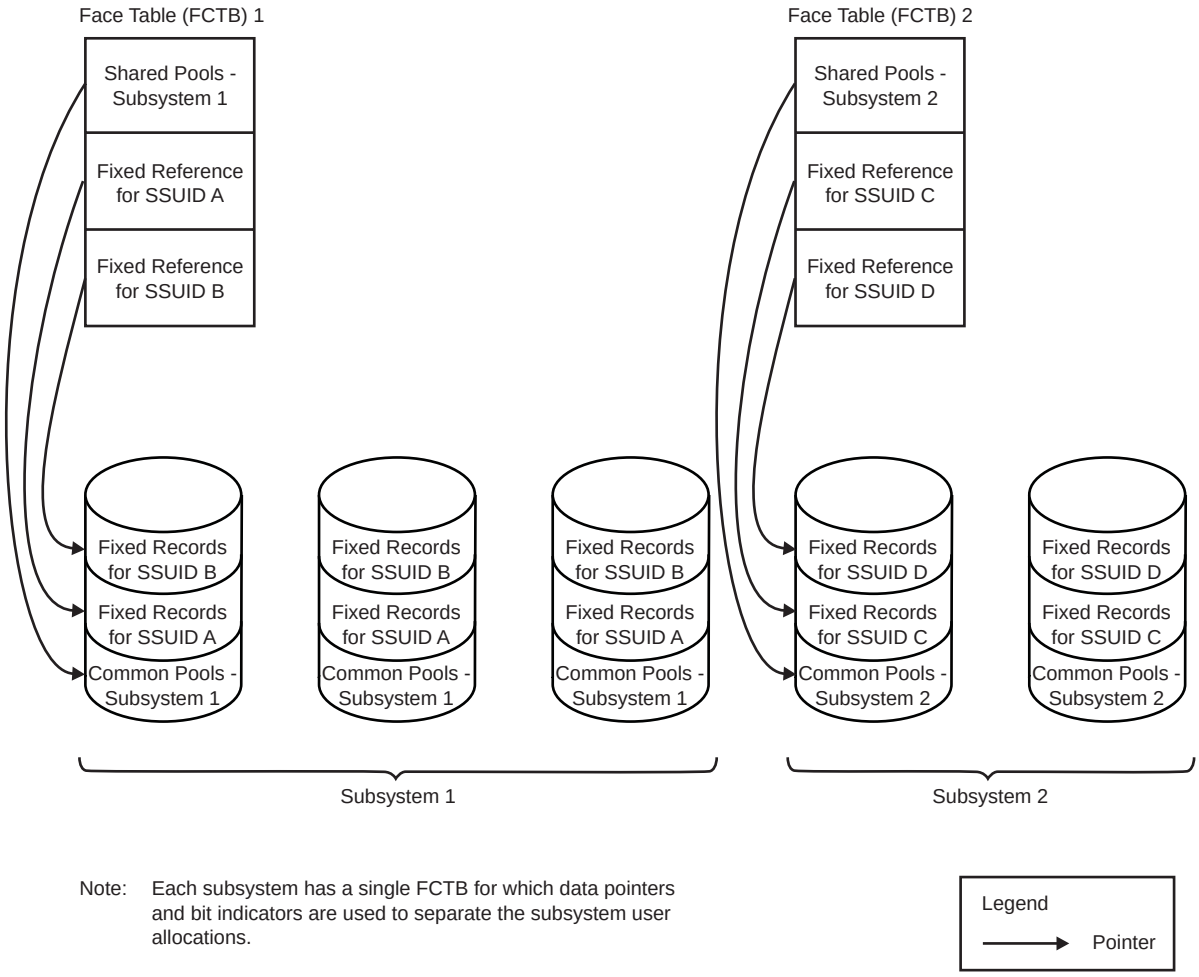


Figure 42. Relationship of FCTBs to Subsystems and Subsystem Users

Record ID Attribute Table (RIAT)

There is one item in the RIAT for each record ID defined within any subsystem, and there is a RIAT for each subsystem.

Module File Status Table (MFST)

An MFST exists for each subsystem in an MDBF environment, and contains the information necessary to identify subsystems.

Routing Control Application Table (RCAT)

Although the RCAT is considered part of TPF data communications, which is discussed in “Data Communications” on page 169, it is necessary to describe it briefly here in relation to MDBF.

To ensure that an input message is routed to the appropriate subsystem for message processing, the routing control application table identifies both the subsystem and the subsystem user associated with an application.

Global Area and Global Records

Although the concept of globals has not yet been introduced, it is necessary to describe it briefly here in relation to MDBF. See “Globals” on page 139, for more information.

There are global records for each subsystem user in each subsystem. Also, there is a unique main storage global area for each subsystem user in a subsystem. Note that the use of subsystem user global areas could require a significant amount of main storage.

Summary of MDBF

MDBF introduces the terms *subsystem* and *subsystem user* with the following key associations:

- The records identified in a FACE table (FCTB) are allocated to a single subsystem.
- A single subsystem can support multiple subsystem users.
- Each RCAT application is associated with only one subsystem user ID, but several RCAT applications can be associated with the same subsystem user ID.

Within a loosely coupled complex, a *database* is considered to be a unique set of fixed and pool record types allocated across a set of modules.

Unique Records and Shared Records

Data records in the TPF system can be shared or isolated in several ways depending on the requirements of the user installation. This is achieved on the basis of record types.

Record types can be specified as shared or unique based on

- Subsystem user
- I-stream engine
- Processor (in a CPC with multiple I-stream engines, processor refers to the CPC).

For the sake of simplicity, unique and shared records are individually discussed in the context of subsystem user, I-stream engine, and processor. However, based on the requirements of the user installation, any combination can be specified, for example, subsystem user and I-stream engine unique.

Shared Records — Subsystem User

In a TPF system generated with MDBF, record types that are defined as subsystem user shared are accessible to all subsystem users within a subsystem; that is, (1) there is one instance of each ordinal number of the record type and (2) all subsystem users must coordinate with each other to gain access to the record. Pool records are examples of record types that are subsystem user shared.

Shared Records — I-Stream Engine

Record types defined as I-stream engine shared are accessible to all I-stream engines within a central processing complex (CPC).

Shared Records — Processor

Record types defined as processor shared are accessible to all CPCs within a loosely coupled complex.

Unique Records — Subsystem User

In a TPF system with MDBF, any fixed record type can be declared to be *subsystem user unique*. Separate sets of records for a record type that is subsystem user unique exist for each of the SSUs of the subsystem. If the record type is not needed by all subsystem users of the subsystem, a set of records does not exist for those SSUs.

The ability to declare subsystem user unique records allows specific processing to be isolated within a subsystem user. The other subsystem users can be doing identical processing in parallel, keeping their data updates in their own sets of records. (See Figure 43.)

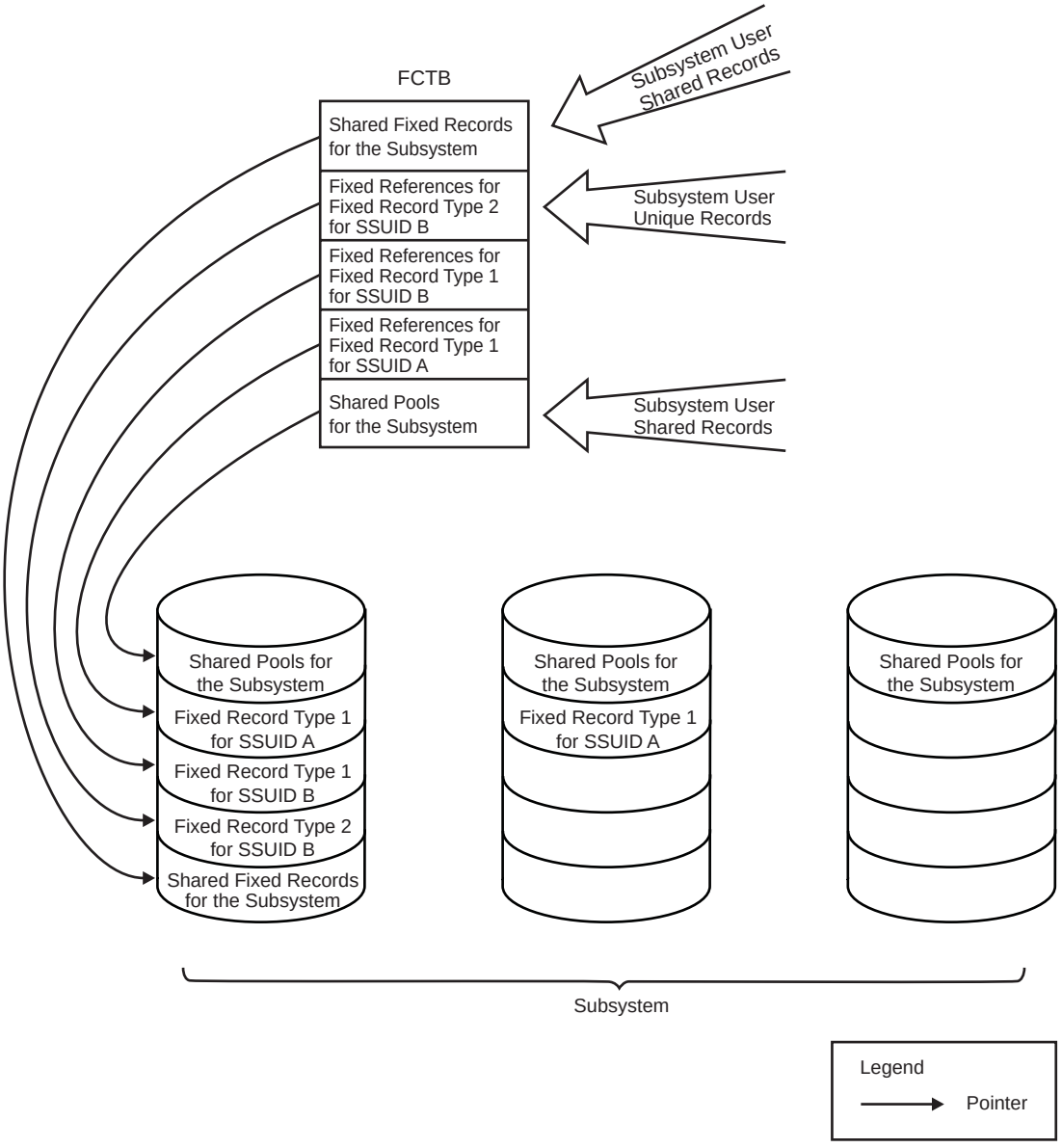


Figure 43. MDBF: One Subsystem, Two Subsystem Users. SSU Unique Records.

Unique Records — I-Stream Engine

Because a TPF system can run in a central processing complex (CPC) with multiple I-stream engines, this same processing isolation is achieved on an I-stream engine

basis by declaring certain record types as I-stream engine unique. See Figure 44 .

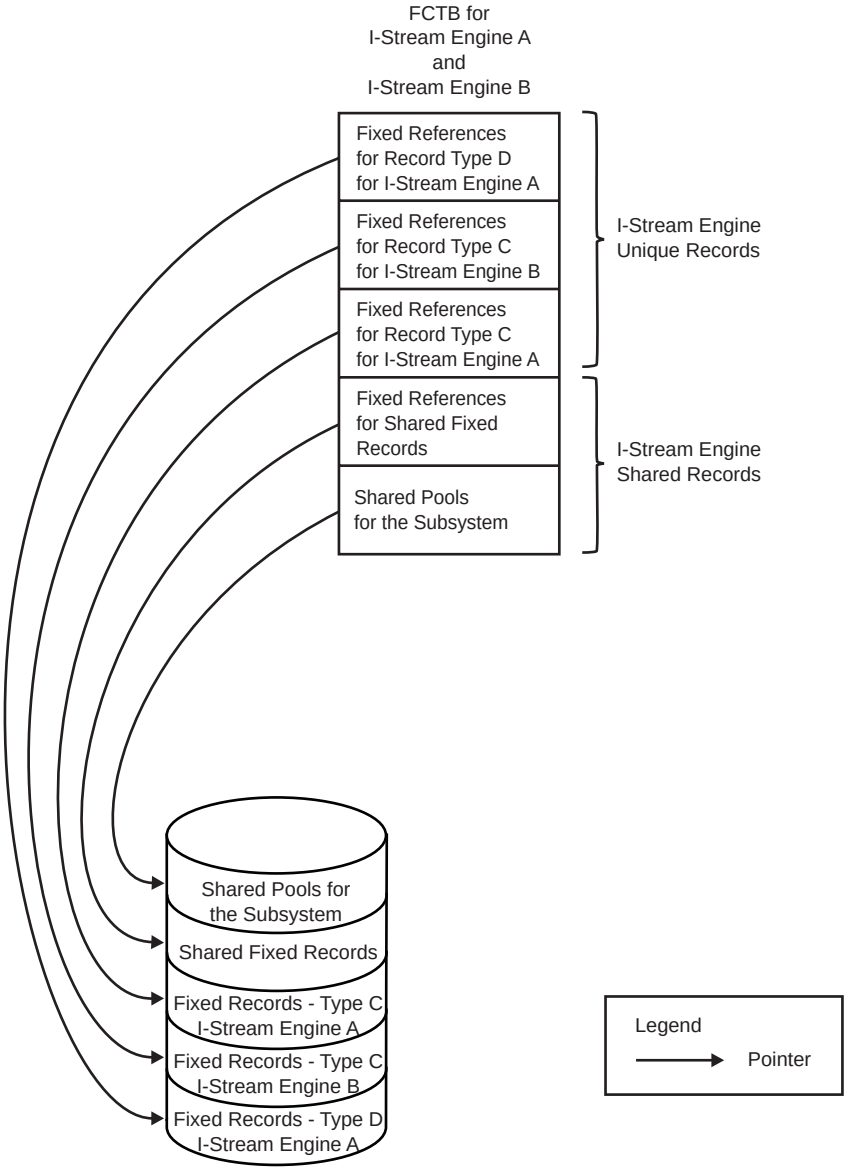


Figure 44. Two I-stream Engines, One Subsystem. I-stream Engine Shared and Unique Records.

Unique Records — Processor

In a loosely coupled complex, *processor unique* records can be used to isolate data updates made by a specific CPC to a record that is only accessible by that CPC. See Figure 45 on page 135.

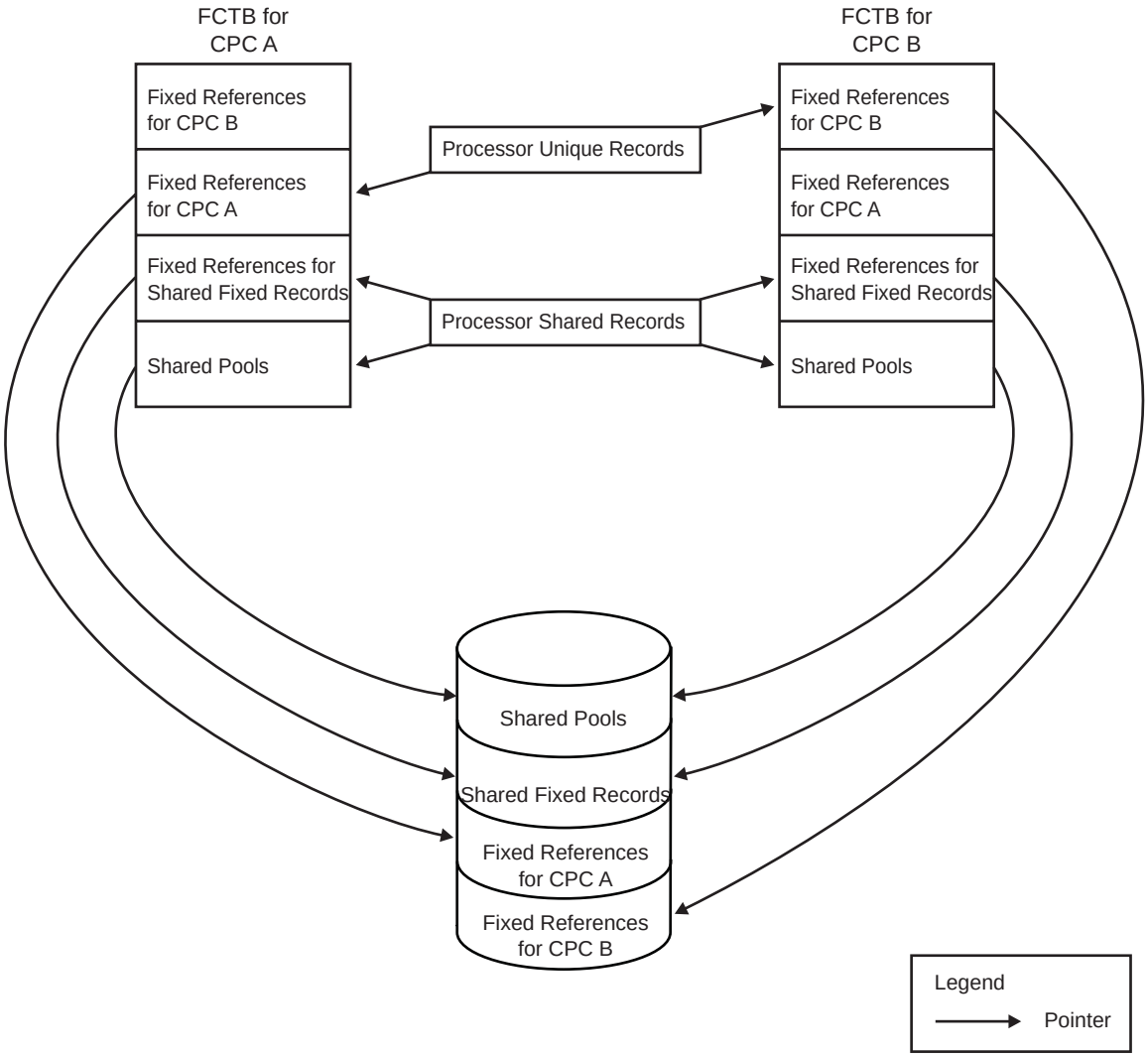


Figure 45. Loosely Coupled Complex. Two CPCs, One I-stream Engine Each.

Basic Subsystem (BSS)

Whether or not the MDBF option is used, the subsystem on which the TPF system programs and associated system records reside is designated as the basic subsystem, which is given the reserved name of BSS. This implies that the TPF system is IPLed and restarted from the basic subsystem. The basic subsystem can be declared with one or more subsystem users if the system is generated with MDBF.

Switch Among Subsystems and Subsystem Users

Some ECB-controlled system utility programs are used for database management and need to access all subsystems and subsystem users in a subsystem. A mechanism to “switch” among subsystems and subsystem users is needed. *Switch*, in this case, means to change an index to a relevant MFST and FCTB used to resolve file references during online execution. See Figure 46 on page 137. Two important cases are:

- A set of ECB-controlled programs must refer to data in some other subsystem user.

  File recoup, a system utility used to reconcile *lost* pool addresses, is an example of such a requirement. Remember, pool addresses are shared among all subsystem users on a given subsystem, so pool address accessing is independent of an SSU ID. However, the specific fixed file records that hold pointers to the pool records can be subsystem user unique.
- The need to process ECB-controlled system programs that are accessible only to the basic subsystem, but are necessary to complete the processing of a set of ECB-controlled programs that are accessible on a different subsystem.

  To appreciate this, think about the problem of an ENTER macro service routine, invoked from an Entry not executing on the basic subsystem, that requests the service of an ECB-controlled program accessible only to the basic subsystem.

  For example, the long message transmission (LMT) program, which resides on the basic subsystem (BSS), must operate on messages whose message content exceeds the size of a small working storage block. That part of the message that is not contained in the storage block is formatted as segments of chained pool records. The pool record addresses are obtained in the subsystem of the program that invokes LMT. An MDBF utility that runs in the BSS is used by LMT. This utility switches to the subsystem of the invoker (so the file reference to the chained pools are resolved correctly) and copies the records by recreating chains of pool records in the basic subsystem. Thereafter, the remainder of the LMT package runs oblivious to the concept of subsystem or subsystem user.

Because file resident ECB-controlled programs are shared, they are accessible to all subsystem users within a given subsystem and are called the *program base* of the subsystem. Also observe that ECB-controlled system programs in the program base of the basic subsystem have the potential to be *shared* among all subsystems. This is different from sharing programs or data within a single subsystem. Shared programs or data within a subsystem are referenced in the subsystem FCTB; no switching is required.

Three fields are included in the ECB to identify the subsystem and subsystem user of an execution environment. System programs use these fields to switch among subsystems and subsystem users.

- Program base identification (PBI) locates the subsystem MFST to resolve file addresses of programs The PBI is needed by the enter macro service routines.
- Database identification (DBI) locates the subsystem FCTB to resolve file addresses of data records. The DBI is needed by the find and file macro service routines.
- Subsystem user ID (SSU ID) is needed to locate a unique set of fixed records identified as unique in the FCTB.

DBI and PBI locate a subset of modules; SSU ID locates a unique reference in the FCTB (see Figure 46 on page 137). So, it is possible to be running the system with a PBI that is different from the DBI; the normal application environment runs with PBI=DBI and an appropriate SSU ID. These fields can be modified with the Cross Subsystem Access Services (CROSC) macro.

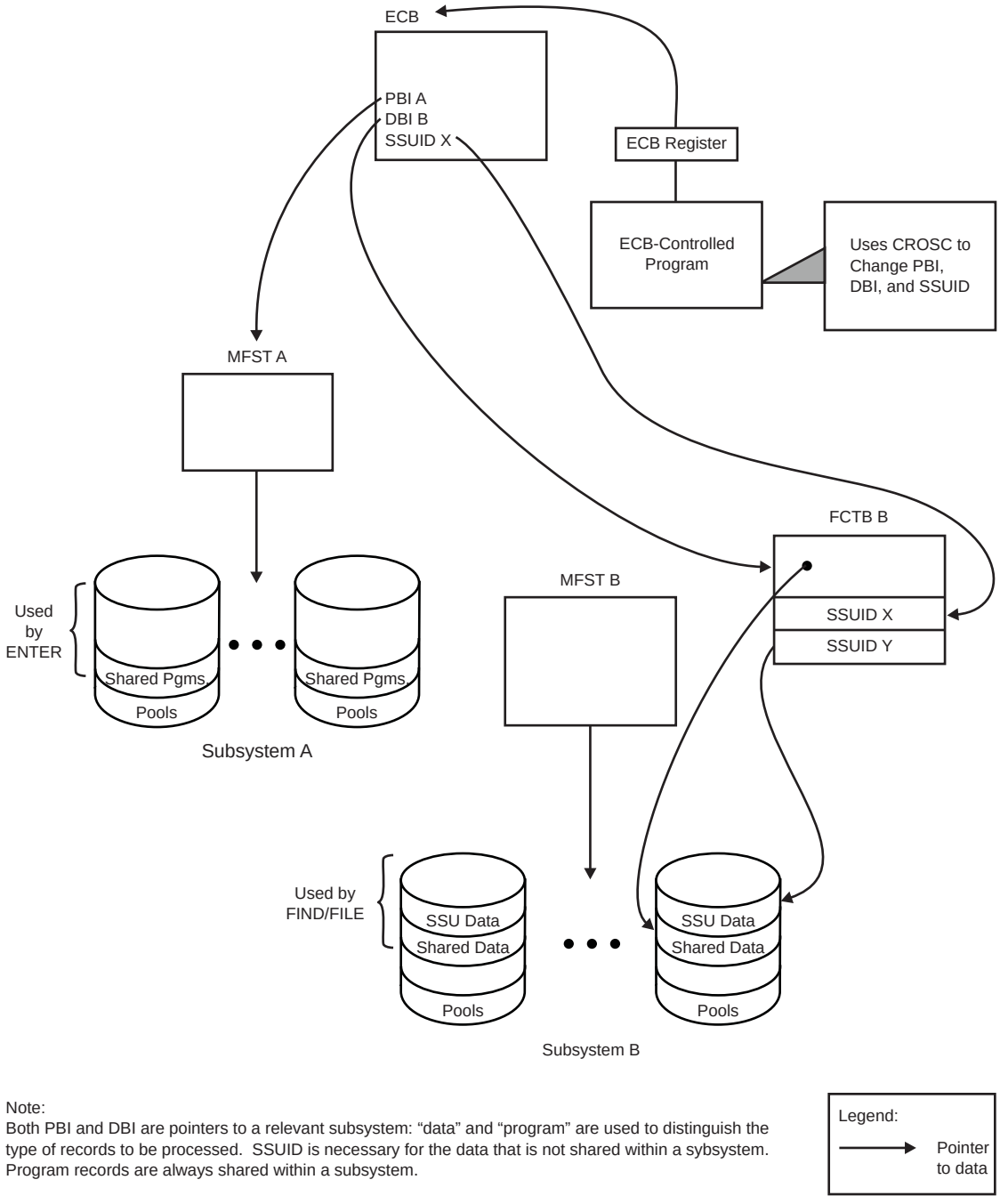


Figure 46. Cross System Access Services

Retain Module Records in Main Storage

In the TPF system, there are techniques used that are ultimately related to the overall objective of performance. Because there is a time delay when accessing data from external storage (such as modules) during which time an Entry is *waiting*, a performance objective is to minimize the waiting time. A way of minimizing the

waiting time is to eliminate it. To this end, there are two mechanisms in the TPF system design that eliminate or minimize I/O accesses:

- Virtual file access (VFA) is the TPF system software disk caching technique for temporarily holding module records in main storage to improve the access time of frequently referenced records by reducing the number of physical I/O operations.

  Any record type (identified by a record ID), fixed or pool, can be identified as a VFA candidate. As with all caching techniques, when the cache becomes full, it is necessary to remove some old records in order to make room for new records. Some basic elements of VFA caching are:

  – Keeping the file addresses of all the records currently in the VFA cache; the lists of these addresses become so large that hashing techniques are employed.
  – Keeping control information for deciding which records to remove.
  – Keeping control information to decide when and if a record in the VFA cache needs to be written to a module. For example, if a record that is about to be removed from the cache has been modified, the record must be placed on a module before disappearing from the VFA cache.
- The TPF global area is a portion of main storage allocated to application program records that, in principle, is the equivalent of records permanently held in VFA cache.

  However, the main storage in which global records reside is distinct from the VFA cache, and the mechanism for accessing global records is through two main storage global directories and associated macro instructions rather than with find and file macros, as is the case for VFA accesses.

  The global records retained in main storage have counterparts on a module, primarily to restore the main storage record when the system is restarted. In contrast to VFA records, records in a global area never need to be removed from main storage because they are permanently allocated. However, global area records that are modified must be written to a module if the modification is to be preserved over a system restart.

The term *caching* has been used in this introduction to emphasize that the TPF support of VFA is a specific instance of a general technique that occurs in other operating systems as well as in other components of the TPF system software and hardware. Within the other TPF system publications the phrase *VFA buffer* is used instead of VFA cache; *VFA buffer* is used hereafter.

The evolution of the TPF system to support both forms of multiprocessing (loosely coupled and tightly coupled) has complicated the procedures used in support of VFA and global records. MDBF adds additional complexity.

Virtual File Access (VFA)

Records are identified as VFA candidates in the record ID attribute table (RIAT). If the records associated with a record ID are identified as VFA candidates, one of two procedures must be specified for writing the record to a module in the event the record is modified. Of course, neither procedure is invoked if the record is read-only, because the procedure is triggered only on a file type macro or on a find and hold that generally implies a file request is expected to follow. However, there is no indication in the RIAT to indicate whether or not records are read-only. And read-only records are only a notion imposed by application design, not by the TPF system. So, an application is free to update a read-only record. Therefore, one of the following attributes must be specified despite the intended use of records:

- Delay file — When a file type macro is issued, the record is ***not*** written out to a module ***until*** one of the following conditions occurs:
  - The record is not currently accessed by one or more Entries and the space in the VFA buffer is needed for another record (in the TPF system, this is called the aging out process).
  - Operation of the TPF system is changing modes (this is an operator action); in the TPF system, this is referred to as cycle down.
  - A catastrophic software error occurs and the TPF system error recovery process determines that either an IPL is necessary to restore the system to an operational state, or a switchover to another CPC is required. If a switchover occurs, the operator initiates the IPL, and this is referred to as a hard IPL. When the IPL is initiated by the error recovery process, it is referred to as a software IPL.
- Immediate file — Writing to a module occurs whenever a file type macro is issued. At some performance penalty, this keeps the module copy more current (than a record that is delay-filed) in the event of unplanned system restarts.

Although the VFA attributes are kept in the RIAT, record sharing tables (RST) are used to keep the information necessary to control the VFA buffers of main storage, such as the reference bits, file addresses, subsystem IDs, and subsystem user IDs.

VFA is always active; in earlier versions of the TPF system, the use of VFA was optional.

User Exit for VFA

Indicating *user exit* within the RIAT permits an installation-supplied program to access and change the VFA attributes in the RIAT. *User exit* is a general facility of strategically placed locations in the system where control is given to programs supplied by an installation. One such exit is located in the system support of VFA. If *user exit* is specified for the records associated with a record ID, an installation-written program receives control at the designated location in the VFA system code, whereupon modifications can be made to the VFA attributes in the RIAT. The installation programs reached through user exits are, in essence, extensions to system programs and must be implemented and used with caution.

Considerations for a Loosely Coupled Complex

The limited lock facility (LLF) provides an efficient locking mechanism for records resident on modules among loosely coupled CPCs. However, LLF does not provide commands that permit synchronization of modifications to records resident in VFA. This limits the set of records that can be resident in VFA in a loosely coupled complex. (LLF provides locking but not synchronization.)

If the multi-path lock facility (MPLF) or coupling facility (CF) is available exclusively, the TPF system supports the synchronization of such a modification to corresponding records in the VFA buffers located in the main storage of the different CPCs in a loosely coupled complex. If a record that is located in the VFA buffers of several CPCs and defined as a VFA synchronization candidate is modified on one CPC, the other CPCs are then notified that an update is being made.

Globals

The TPF system provides an efficient mechanism for accessing data and communicating between application programs, resulting in quick response times even during high system activity, by avoiding delays caused by I/O operations. This mechanism is known as globals, and consists of areas of main storage allocated for use by applications. Typical uses are:

- Passing data or other information between application programs to avoid I/O operations
- Providing ready access to frequently referred to information, such as fare data in an airline application.

The TPF system provides the GLOBZ macro for applications to access and to possibly modify globals. To preserve the integrity of data in globals, they are also maintained on file storage. Keypointing is used to ensure that data that is modified in the global area is reflected in the main storage global area after a system restart.

The following is a very brief description of globals and concepts within the TPF system. See Figure 47 on page 141.

Globals within the TPF system are contained in fixed locations in main storage. These locations are identified as global areas (see global areas 1, 2, and 3 in Figure 47). Within the main storage global areas, and also residing on a module (as global blocks), is a logical collection of data known as global records. These records can have various attributes (such as keypointability, SSU uniqueness or commonality, and I-stream uniqueness or commonality). A global record can be subdivided into global fields, which range in size from 1 to 256 bytes, are individually addressable using the GLOBZ macro, and can share many of the attributes of global records. A global directory is used to manage TPF globals. It is a table containing pointers to the main storage and file addresses of global records in global areas 1 and 3.

Globals that reside below the 16-MB boundary and are, therefore, accessible to programs running in 24-bit addressing mode are the primary globals. Extended globals reside in an area of storage defined above the 16-MB boundary. The extended global area greatly increases the amount of main storage available for use as globals. For each primary global area there is a corresponding extended area. A system can be generated with or without extended globals.

Global synchronization is necessary for communication among active I-stream engines in a central processing complex (CPC) across CPCs in a loosely coupled (LC) complex to maintain currency after modifications have been made to global fields and records. Although the TPF system does not automatically perform global synchronization, the SYNCC macro is provided for applications to perform this task. In a CPC with multiple I-stream engines and also in an LC environment, the system interprocessor global table (SIGT) contains the information necessary for controlling the locking, unlocking, and synchronizing of synchronizable globals.

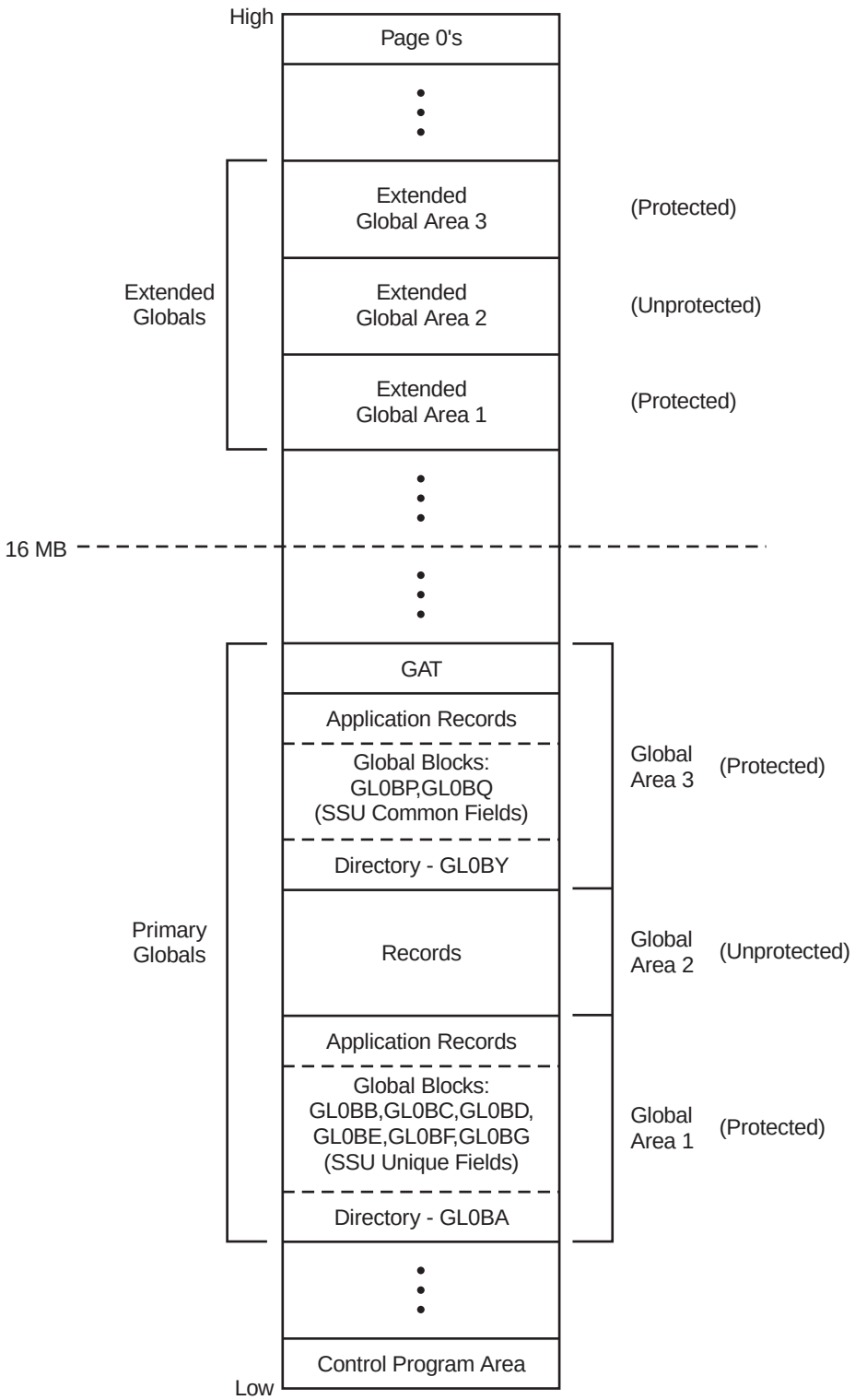


Figure 47. Global Storage Allocation for a TPF Basic Subsystem with a Single I-Stream. Terms labeled GL0xx correspond to the names of assembler DSECTs associated with the areas.

MDBF Considerations

Within a TPF subsystem, a separate global area can be assigned to main storage for each subsystem user. See Figure 48 on page 142. This implies that names of the global areas are identical for all subsystem users because the programs within

a subsystem are shared, and references to the global fields must therefore use the same global names. However, the content of the fields of the same names within the main storage global areas unique to each subsystem user is different during online execution.

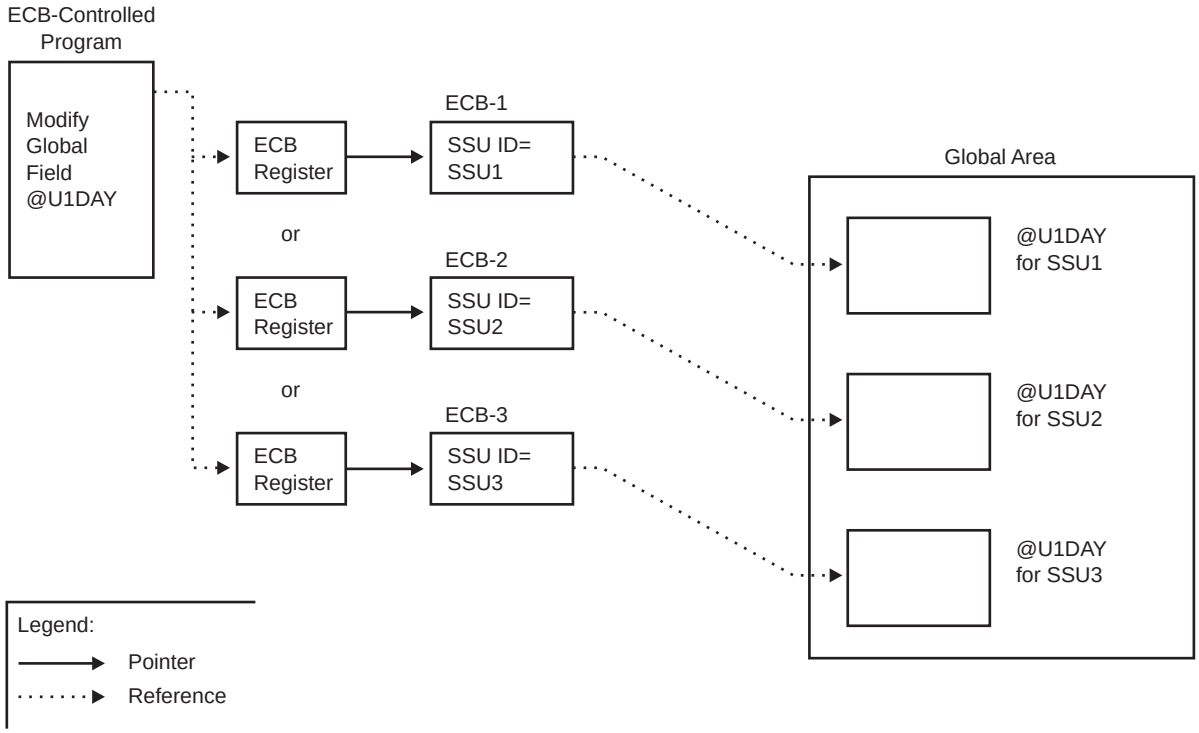


Figure 48. Globals. MDBF Considerations

Multiprocessing Considerations

Some records in the global area, shared by all CPCs, are written to a module in order to synchronize shared global records held in the private main storage of several CPCs within a loosely coupled complex. The use of a module for this synchronization also has the effect of keypointing; in this case, processor shared records are written to file storage.

No XLF locking or record synchronization is necessary on keypointable global records, because these records are always processor unique.

Retain Module Records in Module Cache Memory

In addition to software system features designed to achieve the performance objective, the TPF system also takes advantage of hardware facilities to meet this end. The module record cache is a hardware facility that is an integral part of a module control unit (CU) (also called a module controller). Note that not all module controllers used in a TPF system are equipped with this facility.

Based upon a user installation's requirements, the TPF system can be configured with:

- All module caching controllers
- No module caching controllers
- A combination of both types of controllers.

Module record caching is similar to VFA caching with regard to:

- The use of intermediate storage between the application and the hardware
- The use of the record ID attribute table (RIAT) to specify caching attributes
  - Candidacy characteristics
  - The type of write to be performed.

However, there is an important difference: data records that reside in the module cache allow multiple CPCs in a loosely coupled complex to share a single copy of the data, whereas, data records that reside in VFA cache are usually processor (CPC) unique data records.

Records that are not defined as module record cache candidates are written directly to the module surface, and the use of the module record cache is not called. Records that are defined as module record cache candidates are first written to the cache before ultimately being written to the module surface.

There are two general categories of module record caching: fast write and retentive write. The main difference between fast write and retentive write is the point at which the TPF system is notified that the I/O command is complete.

- Fast Write — the data is written to the cache, and the TPF system is notified of its completion at this point. The data is ultimately written to the module surface. The fast write caching attribute is similar to VFA delay-file.
- Retentive Write — the data is written to both the cache ***and*** the module surface before the TPF system is notified that the operation is complete. The retentive write caching attribute is similar to VFA immediate-file.

General Data Sets

A general data set provides a data interface between offline and online components. Data is either created online (for example, management reports) to be processed by offline programs or data is created offline (for example, a fare database for an airline application) for online processing.

A TPF general data set is similar to an IBM MVS data set and, likewise, is identified by a data set name. The records of a data set are allocated sequentially within a module (called *volume* in an IBM MVS system). See Figure 26 on page 98 for more information about vertical allocation. A general data set can span modules.

Records in standard TPF record sizes are accessed online by using the find and file macro requests. Records that are not restricted to the standard TPF record sizes are accessed online by the file data chain transfer (FDCTC) macro.

Entries refer to the data contained in a general data set by requesting a specific data set name.

General Files

A general file is a sequentially organized set of data that is created by TPF offline programs running under the IBM MVS system. The data is then processed online.

In principle, a general file is similar to a general data set. The primary difference is that the data on a general file is created using non-standard IBM MVS access methods, whereas, when data is created offline for a general data set, standard IBM MVS access methods are used.

Loosely Coupled Multiprocessing — A Database Perspective

Loosely coupled multiprocessing was incorporated in the TPF system to satisfy performance requirements; that is, the need to process more messages for each second, where most of the messages imply access to the same database, as in airline reservation systems. In a loosely coupled complex, each CPC has private main storage but all share the same module control units. Loosely coupled support provides additional MIPS for executing the shared set of application programs. Subsequently, tightly coupled multiprocessing was incorporated for the same reason; that is, to increase processing capacity. In both cases, the TPF system structure was influenced by the goal to permit multiple CPCs to process identical sets of application program segments and access a shared database.

Loosely coupled multiprocessing uses the external lock facility (XLF) to lock records that prevents data records from being updated simultaneously by multiple CPCs in a loosely coupled complex. These locks are set and interrogated by *find and hold*, *find and unhold*, and *unhold* macro service routines at a negligible impact to performance.

Record Hold Table and XLF Lock Table

To process a find and hold request in a loosely coupled complex, the TPF system first checks the main storage record hold table (unique to a CPC) to see if the record is already held by this CPC. If it is held, the request is queued; there is no need to check the XLF lock table until the record is unheld within this CPC. If the record is not held by this CPC, the TPF system holds the record for this CPC and checks the XLF lock table.

- If the record is not locked in the XLF lock table, the lock gets set and the record is given to the requesting Entry
- If the record is locked in the XLF lock table (it is already being used by another Entry in another CPC), XLF ***remembers*** this request, and the request by this CPC is queued until XLF informs it that the lock is now available.

The reasons for maintaining a main storage record hold table are:

- This table can be used to restore locks in the XLF lock table during error recovery procedures.
- The main storage record hold table contains more information than the XLF lock table, such as pointers to Entries.

Database Utilities

A large volume of application data records organized to enhance performance are usually characteristics of a TPF environment. These characteristics dictate the need for special support programs that are often referred to as database utilities.

Database utilities provide a variety of functions:

- Ensure against permanent loss of data by copying online data from a module to magnetic tape and, when necessary, copying the data back to the online module (file capture and restore)
- Facilitate the expansion of file storage (database reorganization)
- Maintain file storage integrity (file copy)
- Recover long-term pool storage (file recoup)
- Maintain pool storage (pool directory maintenance)
- Initialize the online pool database (pool directory generation).

While most of these utilities are run on a regular basis, the utilities used in the initialization process are generally run only once (to generate the system) or very rarely (when the system definition requires change).

Generally, utilities are run during periods of low system activity and, in a loosely coupled environment, on only one CPC. They are also sensitive to subsystems and subsystem users.

The complexities of multiprocessing and MDBF are ignored in this conceptual description of the database utilities.

File Capture and Restore

The constant availability of the online data increases the exposure of file storage to the effects of software and hardware malfunctions. This exposure is minimized by ensuring that critical data can be replaced if necessary.

Maintenance of copies of file storage on auxiliary storage media (such as other module devices and magnetic tape) helps to protect against the loss of critical data. The process of copying file storage to auxiliary storage is called file capture or simply capture, and the process of restoring the data to the file storage is called restore.

The TPF system provides the ability to tailor the restoration of data to the severity of the loss of data. A full restore restores all file storage, whereas a partial restore restores one or more devices. Partial restores occur more frequently than full restores. The principal reason for a partial restore is the physical destruction of the data on a module.

The TPF system provides an online capture facility that *captures* data records during normal system operation, but during periods of low activity. Each file storage device is copied to magnetic tape. Simultaneously, a separate tape, called an *exception tape*, collects a copy of all records modified after they have already been captured. The combination of these two sets of tapes (capture tapes and exception tapes) is a total representation of the database at the instant that the online capture completes.

By the very nature of its function, the capture utility places very high demands on module resources, which impacts TPF system performance. To avoid over utilization of a group of channel paths or control units, the capture utility performs load balancing based on limits specified by the user installation. This minimizes the impact that this utility has on the I/O throughput of a TPF system.

Full restoration of a system brings the system back to the date and time of the capture completion. Additional programs or procedures may be required to reconstruct the file activity that occurred between the time of capture and the time of restoration. Because the data is application dependent, the additional reconstruction is an application function.

Database Reorganization

Normal system growth usually dictates an increase in the number of file storage devices. The addition of new devices provides an increase in both fixed and pool areas. While the pseudo module support, dispensing techniques, and directory maintenance used for the pool addresses permit the temporary addition of devices to the pool area, a database reorganization is required for the expansion of the fixed area and for permanent expansion to the pool area.

The expansion of the file area is accomplished by a TPF system utility called database reorganization. The database reorganization program collects all records using the record types and ordinal numbers of the current system definition. These records are then written into the new database configuration by using a new FACE table definition. Because of FARF, a database reorganization is transparent to application programs.

Figure 49 shows an example of reorganizing a database from three devices to four devices. Notice that the physical location of each record is changed while the numerical sequence of the records is retained. Records 10 and 11 are highlighted.

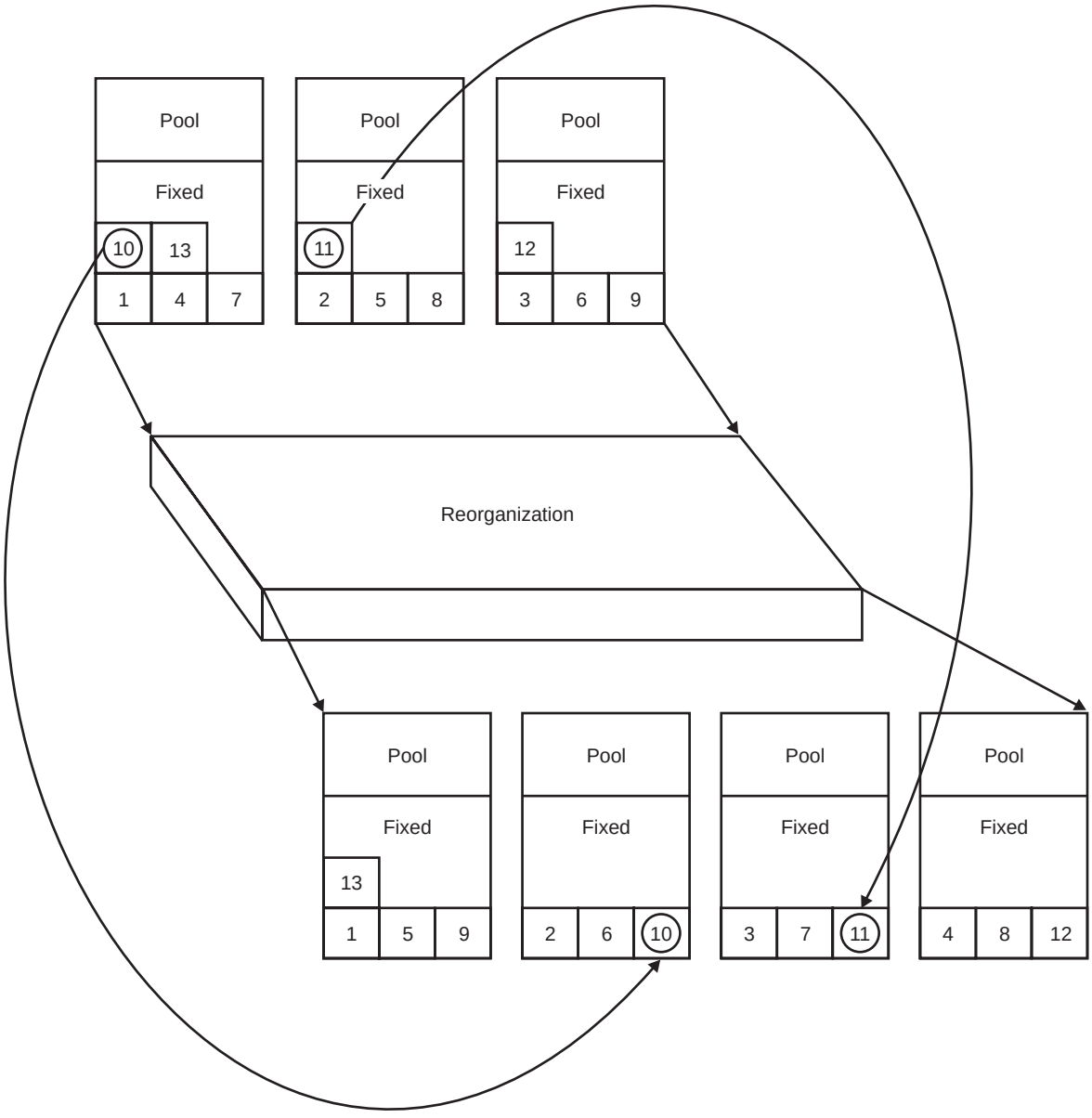


Figure 49. Database Reorganization

File Copy

The TPF system provides the file copy utility as a means of maintaining the module hardware devices by duplicating and replacing in-use modules during system operation. Therefore, the TPF objective of high availability is maintained.

This utility can:

- Copy an in-use module to another module and replace the in-use module with the new module without loss of data or system interruption
- Restore or recreate the duplicate records from a currently online module to the duplicate records to a module that is being brought online.

Although the utility is designed for maximum efficiency, TPF system performance is adversely impacted because of the high demand on the module control units needed for the two modules used by the utility.

In a loosely coupled complex, the copy process is synchronized across the CPCs using XLF.

File Recoup

Occasionally, application programs may fail to return long-term pool addresses to the system, in which case the records are lost for further application usage. This causes the amount of available long-term pool records to be diminished and, if left unchecked, would result in pool depletion. Correcting application errors reduces this problem. Additionally, whenever the TPF system experiences a catastrophic error, some Entries may be partially processed. After a system restart, the TPF system, in order to guarantee data integrity, ***throws away*** a pre-defined amount of long-term pool addresses, which are never returned.

The file recoup utility is provided to recover usable long-term pool addresses. These *lost* addresses are made available to the system for subsequent reuse. The file recoup utility interfaces closely with the application environment to determine which long-term pool addresses are validly in use. A by-product of the file recoup utility is information that identifies application programs which may have lost long-term pool addresses.

Long-term pool addresses are lost in the following ways:

- As a result of errors in application programs, in which programs fail to release file addresses when they should or release file addresses when they should not.
- As a result of the overt action by TPF system programs during a system restart. The time between an update of a pool directory record and an unplanned shutdown cannot be predetermined. However, the file copy of the directory records must be used by system restart programs. This means some long-term pool addresses may have been dispensed and are not accurately reflected in the file copy of the directory record used by the restart programs. Therefore, a pool restart program sets a predetermined number of long-term pool addresses to the unavailable status within appropriate directory records. This ensures that the same long-term pool address is not dispensed to two different Entries causing a certain overwrite of data. This procedure means that some long-term pool addresses are always lost following a system restart.

File recoup reconciles the long-term file pool directories with the actual status of the pool area, and in so doing, provides error summaries that can be used to isolate the causes of directory discrepancies. The file recoup program must access every fixed record and main storage table that can reference a pool record or a chain of

pool records. The data gathered during this record accessing and *chain-chasing* phase is then used to create file pool directories that reflect the actual status of the file pool area. These reconstructed directories in which lost addresses have been recovered (recouped) are called pseudo directories. A pseudo directory is an accurate account of the long-term pool addresses at a discrete instant of time.

For more information about file recoup, see *TPF Database Reference*.

Pool Directory Generation and Maintenance

Use the ZPOOL GENERATION command to create new directory records or change existing directory records to conform to a changed file layout. For more information about the ZPOOL GENERATION command, see *TPF Operations*. For more information about file pool procedures, see *TPF Database Reference*.

Database Generation

The generation of a database is an integral step in the TPF system generation process. A variety of information must be collected manually to initialize the online database. Some information is configuration dependent; for example, the number of types of terminals and workstations supported. Other information is application dependent; for example, the number flights in an airline's schedule. The process of creating the online database is summarized in the following sections.

File Layout

File layout is a planning process and is a part of system generation.

The placement of programs and data records on a module is generally determined by the anticipated frequency of access. To maximize system performance, the most frequently accessed records are placed toward the center of the module, with the least frequently used records placed toward the outside. This tends to minimize the average seek time. The access frequency of most of the data records depends upon the design of the application. Using information from data collection and reduction, the user installation is able to modify the layout of the database to improve performance.

File layout also includes determining the size and location of various file pool types.

Note: This process of file layout is one of the factors that affects the performance of a TPF system. However, when module record caching is employed, its importance is lessened.

File Allocation

Fixed file allocation requires the use of the offline FACE table generation program (FCTBG), which is used to generate the FACE table (FCTB), based on the layout of the database as determined by file layout.

Fixed File Record Initialization

Fixed file records are initialized after using the offline system test compiler (STC) program to build pilot tapes that are used by a TPF program called the online data loader. The header of the records on a pilot tape contains the record type and ordinal number necessary to file (write) the record to the online database.

Disk Module Initialization

TPF disk modules must be initialized using a standard IBM MVS disk initialization program. This program checks for defective tracks, assigns alternate tracks when necessary, and initializes the volume labels according to TPF standards. Disk module initialization occurs as part of the system generation process.

Disk Module Formatting

Following disk module initialization, all online disk modules must be formatted with the TPF formatter program. Disk tracks are formatted for 4KB, 1055-byte, and 381-byte records. This formatting is based on input supplied by the user installation and must be consistent with the FACE table specifications. Disk module formatting is accomplished during the system generation process.

Data Loading

The final step in creating a fixed database is to load the pilot tapes to the online file modules. This function is provided by the online data loader for the TPF system. A system operator initiates a pilot tape load. Each record on the specified tape is read and filed at its appropriate location in the online database.

TPF Database Facility (TPFDF)

The TPF Database Facility is a database manager for applications that run in the TPF system. TPFDF is a product that enhances the application program interface (API) for applications in the TPF system. It is licensed separately from the TPF system.

Up to this point, the discussion of database organization in the TPF system includes many of the features provided by TPFDF on behalf of the application. Prior to the advent of TPFDF, the application itself was responsible for organizing and managing its data.

To increase the productivity of the application programmer, TPFDF provides:

- A logical method of database organization.
- A set of standardized macros to access data (these macros form the API) for use with high-level assembler, C, and C++ programming languages.

  Note: The ISO-C TPFDF library contains the same functions as the TARGET(TPF) TPFDF library.
- Central routines for database access and manipulation.
- Utilities for database maintenance and testing.

An application is no longer sensitive to the physical implementation of the database when TPFDF is used.

In addition to simplifying the access to TPF data, the Distributed Data Architecture (DDA) feature of TPFDF can be used with the TPF Application Requester (TPFAR) feature to provide access to a IBM DATABASE 2 (DB2) database by applications running in the TPF system.

The management of application data structures is centralized through the use of TPFDF and can be performed by a database administrator (DBA). The DBA

concentrates on design of data structures for performance and ease of access so that the application programmer can concentrate on the design and implementation of the application processing.

TPF File System Support

While not totally compatible with Portable Operating System Interface for Computer Environments (POSIX) standards, TPF file system support provides support for storing and operating on information in the form of stream files. A stream file is simply a file containing a continuous stream of data. Examples of stream files are PC files and the files in UNIX systems.

File ownership and accessibility are controlled using the POSIX-compliant structures of user ID, group ID, and access permissions associated with each file in the file system.

Application programs can interact with the TPF file system through TPF file system C functions. See *TPF C/C++ Language Support User's Guide* for more information.

Differences between Stream Files and Database Files

To better understand stream files, it is useful to compare them with TPF system database files. As shown in Figure 50, a database file is record oriented while a stream file is composed of a continuous string of bits. A database file has predefined subdivisions consisting of one or more fields that have specific characteristics, such as length and data type.

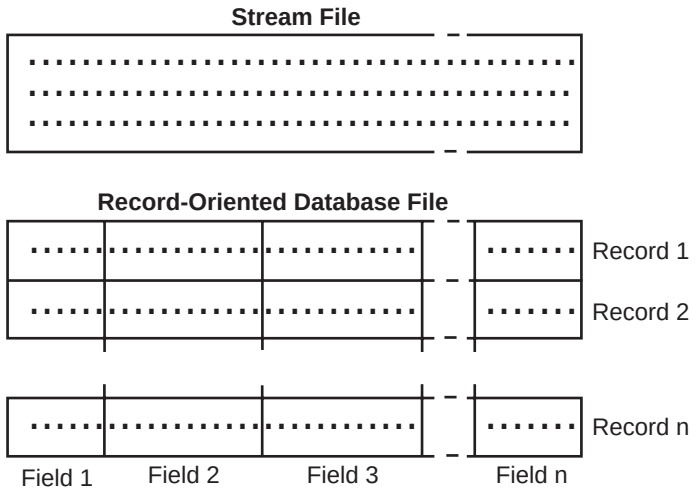


Figure 50. Stream File and Database File Comparisons

The different structure of stream files and database files affects how an application program is written to interact with them and where each type of file is best used in an application program. A database file, for example, is well suited for storing customer statistics such as name, address, and account balance, because these predefined fields can be individually accessed and managed. A stream file is better suited for storing information such as a picture, which is composed of a continuous string of bits representing variations in color. Stream files are particularly well suited for storing strings of data such as the text of a document, images, audio, video, HyperText Markup Language (HTML) documents, Java applets, and other Web-oriented files.

Using Stream Files in Programs

You can create C language programs that interact with stream files by using TPF file system C functions. There are some fundamental differences in the way you operate on database files from the way you operate on stream files.

The differences result from the different structure (or perhaps lack of structure) of stream files in comparison with database files. To access data in a database file, you typically define the fields to be used and the number of records to be processed. To access data in a stream file, you process the entire file sequentially or you indicate a byte offset and a length.

Because you define the format and characteristics of a database file ahead of time, the TPF system has knowledge of the file and can help you avoid performing operations that are not appropriate for the file format and characteristics. With stream files, the TPF system has little or no knowledge of the format of the file. The application program must know what the file looks like and how to operate on it correctly. Stream files allow an extremely flexible programming environment, but at the cost of having little or no help from the TPF operating system.

Directories

A *directory* is a file that is used to locate other files by name. Each directory contains a list of files that are attached to it. That list may include other directories.

TPF file system support provides a hierarchical directory structure that allows users and application programs to identify files in the TPF system. You might think of this directory structure as an inverse tree where the root is at the top and the branches are below. The branches represent directories in the directory hierarchy. These directory branches have subordinate branches that are called *subdirectories*. Attached to the various directory and subdirectory branches are files. A file is located by specifying a path through the directories to the subdirectory to which the file is attached. Files attached to a particular directory are sometimes described as being ***in*** that directory.

TPF file system directory support is similar to the directory support provided in UNIX, which was the model for many aspects of the IBM Disk Operating System (DOS), Windows, and OS/2 file systems. In addition, this support provides features typical of UNIX systems, such as the ability to store a file only once but access it through multiple paths using links. Figure 51 on page 152 shows an example of an hierarchical directory structure.

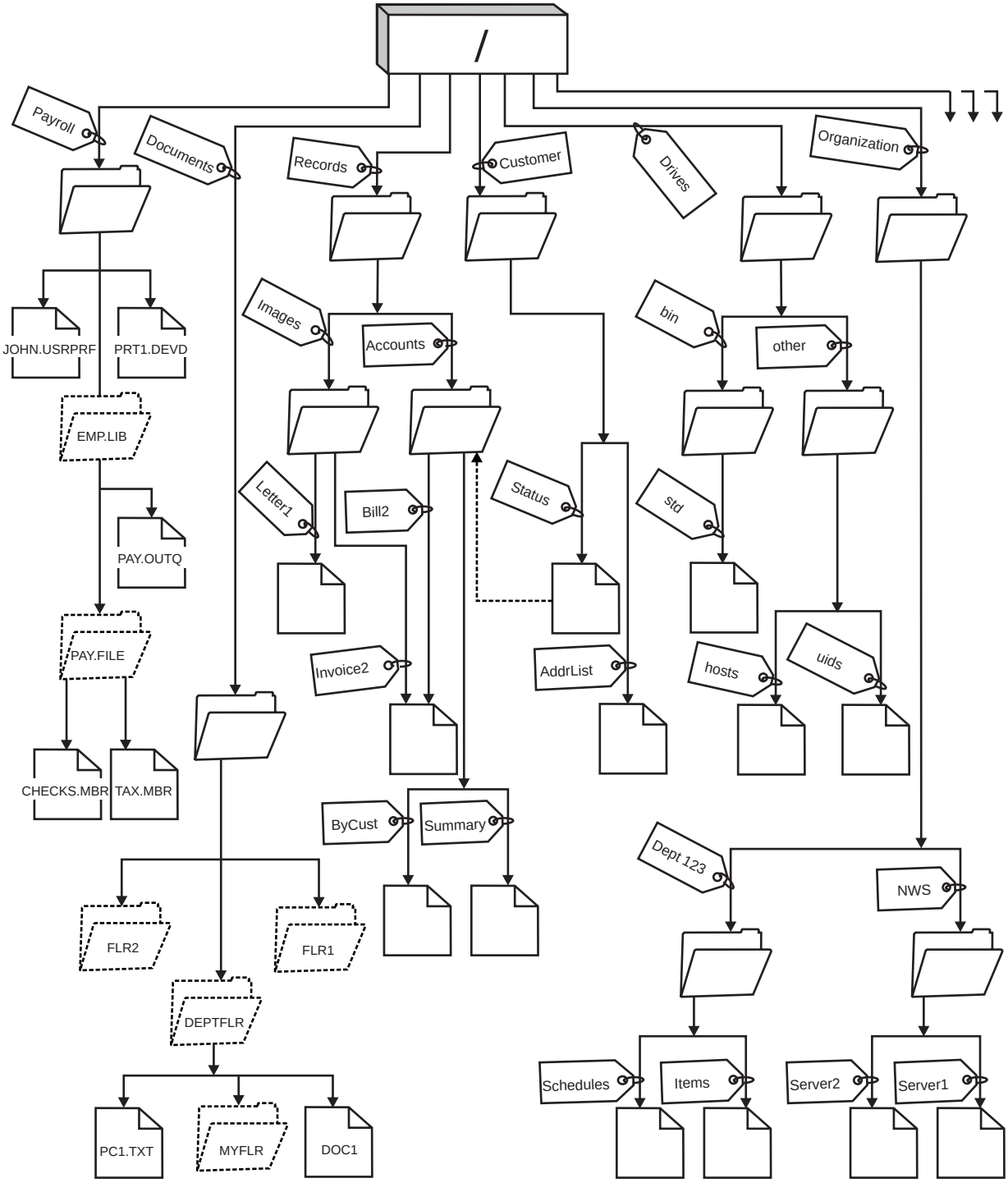


Figure 51. TPF File System Directory Tree Example

Current Directory

When an application program requests an operation on a file the system looks for the file in the current directory unless the application program specifies an absolute path name. The current directory is also called the *current working directory* or just *working directory*.

The TPF_CWD_PATHNAME environment variable is used as the current directory when an application program starts. An application program can specify a directory other than the current directory by calling the `chdir` function at any place in the application program.

Path Name

A path name (also called a pathname) tells the system how to locate a file. The path name is expressed as a sequence of directory names that can also be expressed as symbolic links (see “Symbolic Link” on page 155), followed by the name of the file. Individual directories and the file name are separated by a slash (/) character; for example:

```
directory1/directory2/file
```

There are two ways of indicating a path name:

- An absolute path name (also known as a full path name) begins at the highest level, or *root directory* (which is identified by the / character). For example, consider the following path from the / directory to the file named Smith.

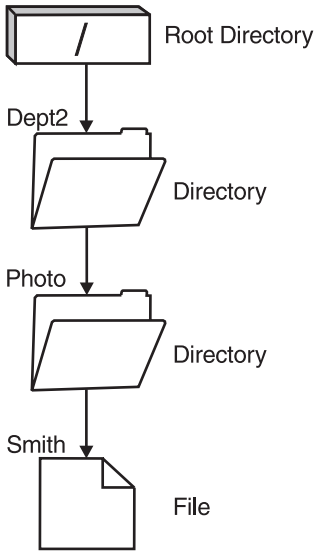


Figure 52. Path Name Components

  The absolute path name to the Smith file is as follows:

  ```
  /Dept2/Photo/Smith
  ```

- If the path name does not begin with the / character, the system assumes that the path begins at the current directory. This type of path name is called a *relative path name*. For example, if the current directory is Dept2 and it has a subdirectory named Photo containing the file Smith, the relative path name to the file for that user is:

  ```
  Photo/Smith
  ```

  Notice that the path name does not include the name of the current directory. The first item in the name is the directory or file at the ***next level below*** the current directory.

Link and Symbolic Link

As shown in Figure 53, a *link* is a named connection between a directory and a file. A program can tell the system where to find a file by specifying the name of a link to the file. A link can be used as a path name or as part of a path name.

It is convenient to think of a file as something that has a name that identifies it to the system. In fact, it is the directory path to the file that identifies it. You can sometimes access a file by giving just the name of the file. You can do this only because the system is designed to assume the directory part of the path under certain conditions. The idea of a link takes advantage of the reality that it is the directory path that identifies the file. The name is given to the link rather than the file.

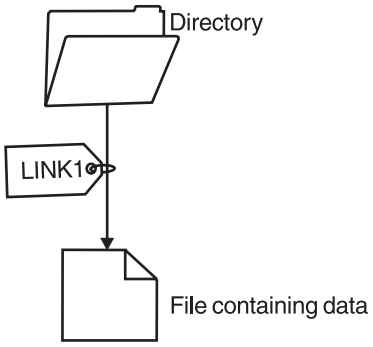


Figure 53. Identifying a File by a Link

There can be multiple links to the same file. For example, two application programs can share a file if each application program has a link to the file.

There are two types of links: *hard links* and *symbolic links*.

Hard Link

A hard link, which is sometimes called a link, cannot exist unless it is linked to an actual file. When a file is created in a directory, the first hard link is established between the directory and the file. Application programs can add other hard links. Each hard link is indicated by a separate directory entry in the directory. Links from the same directory cannot have the same name, but links from different directories can have the same name.

The TPF file system supports multiple hard links to a file either from the same directory or from different directories. The one exception is where the file is another directory. There can be only one hard link from a directory to another directory other than the *dot* (.) and *dot-dot* (..) directory entries. Figure 54 on page 155 shows an example using multiple hard links to a file.

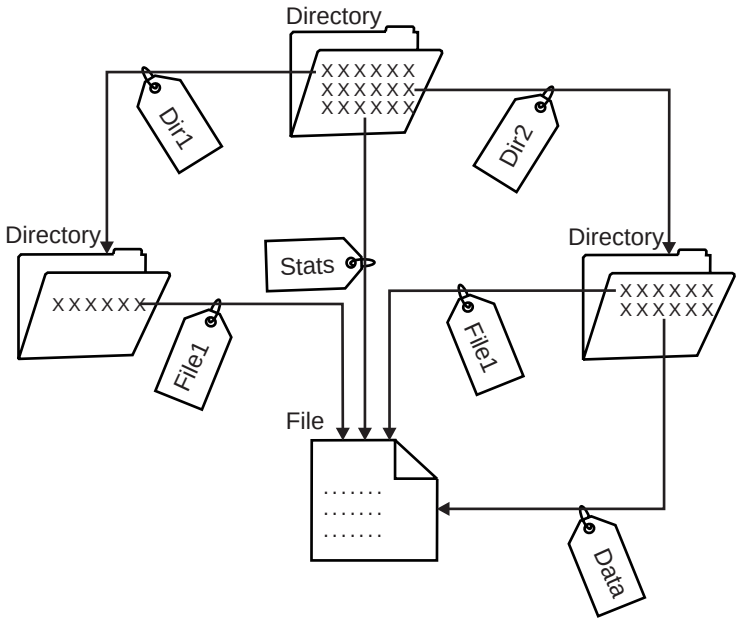


Figure 54. Multiple Hard Links to a File

Hard links can be removed without affecting the existence of a file as long as there is at least one remaining hard link to the file. When the last hard link is removed, the file is put at the end of an available list. If the application program continues to use the file, it can lose the file when the system reuses the storage.

Symbolic Link

A symbolic link, which is also called a soft link, is a path name contained in a file. When the system finds a symbolic link, it follows the path name provided by the symbolic link and then continues on any remaining path that follows the symbolic link. If the path name begins with a /, the system returns to the / (root) directory and begins following the path from that point. If the path name does not begin with a /, the system returns to the immediately preceding directory and follows the path name in the symbolic link beginning at that directory.

Consider the following example of how a symbolic link might be used.

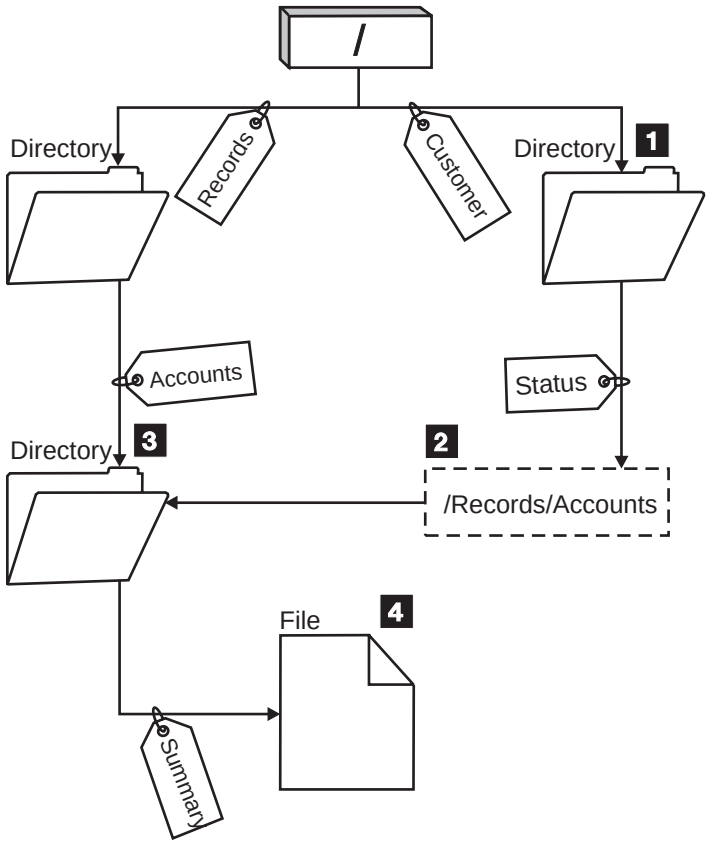


Figure 55. Using a Symbolic Link

A user writes a program to show the status of customer accounts. The program uses the following path name:

```
/Customer/Status/Summary
```

The system follows the *Customer* link, which leads to a directory **1**, and then follows the *Status* link. The *Status* link is a symbolic link, which contains a path name **2**. Because the path name begins with a /, the system returns to the / (the root directory) and follows the *Records* and *Accounts* links in sequence. This path leads to another directory **3**. Now, the system completes the path in the path name provided by the program and follows the *Summary* link, which leads to a file **4** containing the data needed by the user.

Unlike a hard link, a symbolic link can exist without pointing to an existing file. Therefore, you can use a symbolic link to provide a path to a file that will be added later. You can also use a symbolic link as a variable path name that can be changed later.

Hard Link and Symbolic Link Comparisons

When using path names in programs, you have a choice of using a hard link or a symbolic link (see “Link and Symbolic Link” on page 154). Each type of link has advantages and disadvantages. Table 7 on page 157 shows the conditions under which one type of link has an advantage over the other type.

Table 7. Comparison of Hard Link and Symbolic Link

Item	Hard Link	Symbolic Link
Name resolution	Faster. A hard link contains a direct reference to the file.	Slower. A symbolic link contains a path name to the file, which must be resolved to find the file.
File existence	Required. A file must exist in order to create a hard link to it.	Optional. A symbolic link can be created when the file it refers to does not exist.
File deletion	Restricted. All hard links to a file must be unlinked (removed) to delete the file.	Unrestricted. A file can be deleted even if there are symbolic links referring to it.

TPF File System File Attributes

TPF file system files are identified externally by one or more different path names. TPF file system support translates path names to i-numbers, which are ordinals that index into #INODE and #FLOCK records that describe attributes (for example, the access mode) of a file.

Notes:

1. I-number 0 is reserved as the anchor of the TPF file system.
2. I-number 1 is reserved as the root directory of the TPF file system.

Special Files

The TPF system uses special files that can be opened to allow applications access to a device driver. At first glance, special files appear to be files just like any other. They have path names that appear in a directory and they have the same access protection as ordinary files. They can be used in almost every way that ordinary files can be used. However, an ordinary file is a logical grouping of data recorded on disk, while a special file corresponds to a device (such as a line printer), a logical subdevice (such as a large section of a disk drive), or a pseudo-device (such as the null file, `/dev/null`). You will have to create your own special files and device drivers for devices or logical subdevices. By convention, all the files located in the `/dev` directory are special files that correspond to device drivers. The file system initialization program (CBOT) sets up the `/dev/tpf.omsg`, `/dev/tpf.imsg`, and `/dev/null` special files when it initializes the file system. CBOT calls the UBOT user exit, which you can use to set up your own initial directories, files, special files, symbolic links, access modes, user IDs, and group IDs. See *TPF System Installation Support Reference* for more information about the UBOT user exit.

Table 8. Special Files and Their Associated Device Drivers

Special File	Major Device Number	Device Driver Description	Device Address
`/dev/tpf.omsg`	0x0000	Sample code shipped with the UDDWTC segment provides write-only access for output messages that use the `wtopc` function. Use any current modifications of the `puts` and `printf` functions to customize UDDWTC for your own requirements.	The terminal address (LNIATA) in hex (for example, 010000 would refer to the prime CRAS). Therefore, to explicitly open an output message file to terminal address ABCDEF, you would open `/dev/tpf.omsg/ABCDEF`. If there is no location data, the value of EBROUT at the time the file is opened is used as the terminal address.
`/dev/tpf.imsg`	0x0001	Sample code shipped with the UDDIPM segment provides read-only access to input messages that use MI0MI-format core blocks on data level D0. Use any current modifications of the `gets` and `scanf` functions to customize UDDIPM for your own requirements.	None.
None	0x8000	Code shipped with the CDDTBL segment provides support for directories, regular files, and symbolic links.	None.
`/dev/null`	0x8001	Code shipped with the CDDTBL segment provides support for the null device, or dummy file.	None.
`/dev/tpf.socket.file`	0x8002	Code shipped with the CDDTBL segment provides support for mapping socket descriptors to file descriptors, and allows for the use of C application programming interfaces (APIs) that accept file descriptors but not socket descriptors. The file descriptor and corresponding socket descriptor are closed when the entry control block (ECB) exits.	None.
None	0x8003	Code shipped with the CDDTBL segment provides support for pipes. See the `pipe` function in the *TPF C/C++ Language Support User's Guide*.	None.
user-defined file name	0x8004	Code shipped with the CDDTBL segment provides support for FIFO special files (also referred to as named pipes). See the `mkfifo` function in the *TPF C/C++ Language Support User's Guide*.	None.

See *TPF C/C++ Language Support User's Guide* for information about how to add a user-defined device driver.

TPF Collection Support

TPF collection support (TPFCS) is a service for managing the storage and retrieval of data on a TPF database. The data is stored in units known as collections. *Collections* are abstract representations of data having common attributes and functions. Persistent collections maintain their state after the entry control block (ECB) that creates them exits.

TPFCS can be considered an application development tool that integrates database functionality with the application. Potentially complicated data manipulation routines are not needed in application programs because their functionality is already included in the TPF 4.1 system. Furthermore, the TPF 4.1 system does not need to have any knowledge of the format of the data, so more control is given to the application and taken away from the TPF system. TPFCS provides a single, client-level API for storing data elements in a database regardless of the format of the data.

The following table lists topics related to TPFCS and where this information is found:

Table 9. TPFCS Information

Type of TPFCS Information	**Publication**
Application programming	*TPF Application Programming*
Application programming interfaces (APIs)	*TPF C/C++ Language Support User's Guide*
Database information	*TPF Database Reference*
Commands	*TPF Operations*
General overview information	*TPF Concepts and Structures*
Terms and definitions	*TPF Library Guide*
Messages (online, system)	*Messages (System Error and Offline)* and *Messages (Online)*
Migration information	*TPF Migration Guide: Program Update Tapes*

Benefits of TPFCS

TPFCS primarily allows you to facilitate the implementation of problem solutions. This generic benefit can be broken down into the following, and will be discussed in more detail:

- Improving productivity
- Improving application quality
- Improving database integrity
- Controlling recoup using applications.

First, several different abstract data representations are provided, and for each representation, many predefined data manipulation routines are available. Together, all of these routines form a standardized C (C++) library of APIs. As a result, time is not needed on designing new data models and corresponding functions from the beginning. This allows an application to be completed much faster than it would be otherwise. Code is reused over and over, saving time and money.

Second, application quality can be greatly improved by using the TPFCS library. It is less likely that problems will be found with code that has been used several times before than if new code was written. Furthermore, TPFCS APIs hide the low-level TPF interfaces, including ECB usage, data event control block (DECB) usage, and file address information. Because less TPF-specific skills are needed, education time and costs are also reduced.

Next, using TPFCS can improve database integrity. With different collection types (see “Types of Collections” on page 163) and corresponding data manipulation operations available, you are able to find an abstract representation that best matches your data and use a set of consistent APIs with that data. Furthermore, by using this support, it is possible for TPFCS database users to access and retrieve information from the collection without having to retrieve the entire collection.

In addition, TPFCS provides several different locking mechanisms for concurrency control (see “Concurrency Controls” on page 165) to ensure that the database remains in a consistent state at all times. Furthermore, utilities are available for database maintenance and testing. This includes utilities to examine data, repair data that may be corrupted, and capture and restore data.

Finally, TPFCS enables applications to control how data is recouped instead of requiring recoup descriptors to be set up in advance. Applications can create and modify recoup indexes that describe the layout of embedded chain-chase information dynamically.

TPFCS Database

TPFCS is a service that contains all the components that you need to access collections. The TPFCS database consists of user-defined data stores. In those data stores are collections. Each collection consists of elements. Elements can contain character data, binary data, structures, or references to other collections or TPF files.

To relate the concept to familiar terms, you might say that an element could be a record, a collection could be a file, and a data store could be a related set of files. All of these files put together are the TPFCS database. All files that the TPFCS database uses are created as collections and accessed as collections using the TPFCS APIs.

Figure 56 shows the general layout of a TPFCS database:

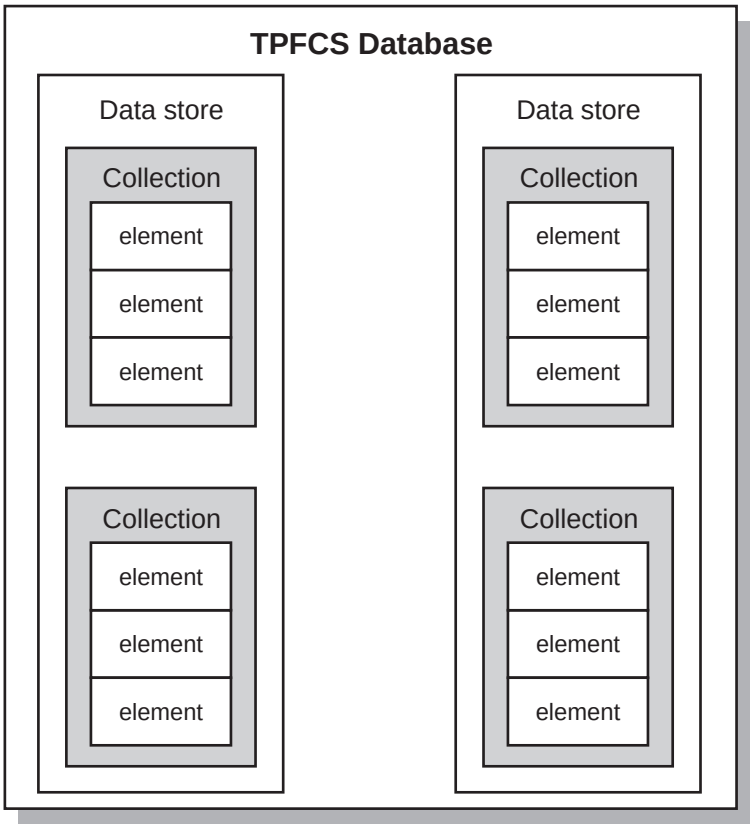


Figure 56. General Layout of a TPFCS Database

Data Stores

A data store (DS) contains collections. Some data stores are created when TPFCS is initialized; others are user-defined and user-named by entering the ZOODB command (see "ZOODB Commands" on page 167). See *TPF Database Reference* for more information about data stores.

Collection Overview

A *collection* is an abstract representation of data that allows you to manage a group of data elements that have common attributes. Collections are used to store and manage elements. Different collections have different internal structures and different functions for element storage and retrieval. See "Types of Collections" on page 163 for information on collections supported by TPFCS.

All persistent collections are assigned an identification token called a *persistent identifier* (*PID*). The PID is architected as a 32-byte token consisting of a format indicator and other information used to locate the collection.

Collection Characteristics: Table 10 describes the four basic characteristics that are used to help you distinguish between the different collections:

Table 10. Collection Characteristics

Characteristic	Description
Ordering	Used when a next or previous relationship exists between elements.

Table 10. Collection Characteristics (continued)

Access by key	Used when a separate key or a predefined part of the element is relevant for accessing an element in the collection.
Equality of elements	Used when you need to test if an entire element value is included in a collection.
Uniqueness of entries	Used when all elements in a collection must have a unique key or value.

Ordering of Collection Elements: The elements of a collection can be ordered in two ways:

- Collections without order have elements that are stored in a random order.
- Collections with order are sorted or sequential:
  - Sorted collections have their elements sorted by ascending binary value. The ordering relation for key sorted collections is determined by key; for nonkeyed sorted collections, it is determined by sort fields.
  - Sequential collections have their ordering determined by either an explicit function that puts an element at a specific position, or by arrival, in which case an add function will add the element as the last element of the collection.

Access by Key: A given collection can have a key defined for its elements. A *key* is an identifier that helps to organize and access elements in a collection. A key can be a data field of the element or it can also be assigned by the application program using some function. A primary key is a value that is separate from the data component of an element (although the same value could be stored in the element). An alternate key can be a data field within the element. Keys let you:

- Organize the elements in a collection
- Access a particular element in a collection with a single function call using a user-defined value.

Key Path Support: There are two types of key paths: primary and alternate. *Primary key paths* are sorted in ascending binary value by the primary key and they can be either unique or nonunique depending on the collection type. *Alternate key paths* support nonunique key path fields (keys) and are sorted in ascending binary value based on the key path field. A maximum of 16 alternate key paths (in addition to the primary key path) can be defined one at a time for each collection.

Key path support enables you to search for and access data in a collection by using the value of a specified data field or key (known as the key path field). TPFCS automatically builds a key path based on the field values, which enables you to access data elements using that specific field.

Key path support provides the capability for TPFCS to support not only the primary, but alternate key paths for the following persistent keyed and sorted collections:

- Key bags
- Key sets
- Key sorted bags
- Key sorted sets
- Sorted bags
- Sorted sets.

Note: Sorted bag and sorted set collections do not have primary keys; they only have sort fields.

For information about using key paths, see *TPF Application Programming*.

Equality of Elements: A collection can have an equality relation defined for its elements. The default equality relation is based on the element as a whole, not just on one or more of its data members (for example, the key). For two elements to be equal, all data members of both elements must be equal. The equality relation is needed for functions such as those that locate or remove a given element. A collection that has an equality relation has element equality.

Uniqueness of Entries: For collections with a key, the terms *unique* and *nonunique* apply to the key; for collections with no key, the terms *unique* and *nonunique* apply to the element. In some collections such as key sorted set, set, key set, and sorted set, two element data components may be equal, but no two keys are equal. Such collections are called *unique collections*. Other collections, including sorted bag, bag, key bag, and key sorted bag can have two equal elements or elements with equal keys. Such collections are called *nonunique collections*.

Types of Collections: There are different collection types, each with predefined functions. Some functions work on all collection types, while other functions only work on a specific collection type.

The following table provides information about the types of collections provided by the collection library and explains some of the key concepts:

Table 11. Collection Type Summary

Collection	Description
Array	An array is an ordered, nonunique collection of elements with no key. Elements can be added but not inserted or removed.
Bag	A bag is an unordered, nonunique collection of elements with element equality and no key.
BLOB	A binary large object (BLOB) is a special array collection of elements with an element size of 1 byte but act as contiguous data. Operations can be performed on a range of elements. A BLOB is also known as a byte array.
Key Bag	A key bag is an unordered, nonunique collection of elements that have a key.
Key Set	A key set is an unordered collection of elements that have a unique key.
Key Sorted Bag	A key sorted bag is an ordered collection of elements that have a nonunique key.
Key Sorted Set	A key sorted set is an ordered collection of elements that have a unique key. A key sorted set is also known as a dictionary.
Keyed Log	A keyed log is ordered by arrival sequence; when the collection is full, the collection will wrap and start overlaying elements at the start of the collection. The elements in a keyed log collection have a key.
Log	A log is ordered by arrival sequence; when the collection is full, the collection wraps and starts overlaying elements at the start of the collection.
Sequence	A sequence is an ordered, nonunique collection of elements with no key. Elements can be added, inserted, and removed.
Set	A set is an unordered collection of elements with unique values, element equality, and no key.

Table 11. Collection Type Summary (continued)

Collection	Description
Sorted Bag	A sorted bag is an ordered, nonunique collection of elements with no key.
Sorted Set	A sorted set is an ordered collection of elements with unique values and no key.

For more information about creating, deleting, and managing collections, see *TPF Application Programming*. For examples of each collection type, see *TPF Database Reference*.

Collection Access Support: The TPFCS database provides a set of services that allows collections to be created, accessed, stored, updated, and deleted. The TPFCS database can handle any collection from 0 to 2 147 483 648 elements in size. The TPFCS database handles the collection as a binary byte string. The collection size is limited only by the available amount of DASD 4-KB long-term pool space. The TPFCS database relies heavily on TPF long-term pool records.

The TPFCS database requires a single TPF record type, which is used to contain the actual collection store directory.

Data Definitions: A data definition (DD) is an integral part of a collection definition. A collection is created using a DD defined for a particular data store (DS). A DD provides attributes for collections in a DS. They are also used to assign record IDs (RIDs) to collections and to specify shadowing (see “Shadowing” on page 166).

See *TPF Database Reference* for more information about data definitions.

Properties: The TPFCS database supports a property service for persistent collections that are created. *Properties* are essentially typed named values that you can dynamically associate with an already existing persistent collection. Once these properties are defined, they can be named, their values can be set and obtained, their access modes can be set, and they can also be deleted. See *TPF Database Reference* for more information about properties.

Collection Lifetimes: TPFCS provides the following collection lifetimes:

Persistent long-term
Exists beyond the life of the creating ECB, resides on DASD in 4-KB long-term pools, and can survive a re-IPL. TPFCS reuses its own long-term pool addresses so applications do not flush through pool records too quickly and cause the TPF system to run out of long-term pool records.

Persistent short-term
Exists beyond the life of the creating ECB, resides on DASD in short-term pools, and can survive a re-IPL. Because the collection resides in short-term pools, it will be deleted when the short-term pools are recycled.

Temporary
Resides in the private heap area of the ECB, overflows to DASD in short-term pools, and cannot survive a re-IPL. It is deleted when the ECB exits because the private heap area of the ECB is reclaimed by the system when the ECB exits.

Cursors

A *cursor* is an internal structure that is used to reference an element in a collection. Cursors are used to iterate through collections and to establish locks on the target collection. Cursors can be used with any collection. Using a cursor provides several advantages:

- It enables easy movement through elements.
- It provides a mechanism for locking entire collections.
- It shortens the code path length for accessing elements.

To use the collections effectively, you often need cursor functions, iterator functions, or both. Cursor functions are used to reference an element in a collection. Iterator functions let you iterate over all elements of a collection and, for example, apply a certain function to all elements or iterate over the collection until you have found a certain element.

Cursors are also used to prevent concurrent updating of a collection while you are accessing elements in the collection.

For complete information about using cursors, see *TPF Application Programming*.

Database Integrity

This section provides information about TPFCS services that maintain database integrity.

Database Access

There are no restrictions on which users can access the TPFCS database. Database access is denied only if the data store you requested does not exist.

Concurrency Controls

TPFCS provides three levels of concurrency control:

1. None (using nonlocking cursors)

   This type of concurrency control uses a nonlocking cursor. The cursor is used for read-only operations on elements in a collection without creating any type of interlock on the target collection. Because the information can change underneath the cursor, it is considered a *dirty-read* cursor.

2. Optimistic (using update sequence counters)

   As a form of database integrity for those collections that allow element updating, an update sequence counter is stored in each element in a given collection and optimistic concurrency is enforced on the element during update processing. *Optimistic concurrency* is a way of controlling database access; a user can read and update a collection element without exclusive access to the collection.

3. Pessimistic (using locking cursors)

   Pessimistic concurrency uses a locking cursor to create an exclusive lock on the collection. Write operations are only allowed by the ECB creating the locking cursor, and requests by other ECBs to create a lock or write data will be delayed until the locking cursor is deleted.

For more information about concurrency controls, see *TPF Application Programming*.

Dirty-Reader Protection: TPFCS provides dirty-reader protection by using the FILNC macro. FILNC will write all new or updated records to DASD before any referencing record is written. This is done to ensure that another user who is attempting to read the collection using a nonlocking cursor will be able to follow a

whole chain. If the updates were filed in the wrong order, it could then be possible for the reader to follow a chain with either holes in it or the chain could point to records that were not even part of the collection.

For more information about the FILNC macro, see *TPF General Macros*.

TPF Transaction Services

TPFCS uses TPF transaction services to establish a commit scope on behalf of the caller for all update requests with the exception of cursor update requests. The application is responsible for commit scopes when using cursors. TPF transaction services will then either commit or roll back the change depending on the success of the request.

Note: To determine which TPFCS functions use commit and rollback protocols, see the *TPF C/C++ Language Support User's Guide*.

Shadowing

TPFCS provides an option when a collection is created that allows you to specify that the collection is to be *shadowed*. When TPFCS shadows a collection, two copies of the collection are maintained, the prime and the shadow. When using normal TPF duplicate records for the collection, shadowing means that TPFCS is actually maintaining four copies of the data.

Validation

TPFCS validation involves performing a check of collection structures to ensure that they are built correctly. Validation is used with the reconstruction function to detect, isolate, and correct internal structural errors.

Reconstruction

Reconstruction involves rebuilding a control record of a collection and the chains that the collection anchors. Such reconstruction would be necessary if data becomes inaccessible as a result of data or control record corruption, logic errors, I/O errors, or a user error.

Database Archives

This section provides information about TPFCS database archive services.

External Device Support

External device support and archiving provide interfaces that will allow you to read and write data from external devices such as tape, general data sets, communications, and any other devices supported by the TPF system.

Archiving

For archiving, the application calls the `TPFxd_archiveStart` function instead of the `TPFxd_externalStart` function. The `TPFxd_archiveStart` function will return a token that the application passes to TPFCS as a parameter on the `TO2_capture` or `TO2_restore` function.

Capture and Restore

TPFCS provides functions to write collections to external storage and retrieve collections from external storage. By using these TPFCS functions and the archiving support of the external device support, collections will be able to be moved in and out of archived storage.

See *TPF Database Reference* for more information about external device support.

TPFCS APIs

The provided functions (APIs) that make up the collection library are divided into three types:

- Collections
- Cursors
- Auxiliary.

See *TPF Application Programming* and the *TPF C/C++ Language Support User's Guide* for more information.

Environment Block

A TPFCS environment block must be created by each application to access a data store. A pointer to that block must be passed on every subsequent TPFCS function call.

Error Handling

When TPFCS finds an error attempting to process a function, it sets an error code in the environment block and returns a `0` to the application program. The application can interrogate the environment block to determine the exact error and retrieve the associated text.

See the *TPF C/C++ Language Support User's Guide* for information about all of the supported TPFCS APIs.

Maintaining TPFCS

This section provides information about the commands used to initialize and maintain the TPFCS database.

Note: Most of the functionality achieved by using the commands can also be implemented by writing applications that use the TPFCS APIs.

ZOODB Commands

The ZOODB commands are used to initialize the TPFCS database, define, change, and display a data store or data definition, and delete a data definition.

The following describes the ZOODB commands:

Function	Description
ZOODB CHANGE	Changes a data definition or data store.
ZOODB DEFINE	Defines a data definition or data store.
ZOODB DELETE	Deletes a data definition or data store.
ZOODB DISPLAY	Displays a data definition or data store.
ZOODB INIT	Initializes support.
ZOODB MIGRATE	Migrates a data store.
ZOODB RECREATE	Re-creates a data store.
ZOODB SET	Sets on or sets off the method trace table or set dump creation on a `TO2_getErrorText` function call.

For more information about the ZOODB commands, see *TPF Operations*.

ZBROW Commands

TPFCS provides browsing support that allows classes, methods, and collections to be located, interrogated, validated, displayed, and dumped. This support is provided by using the ZBROW commands and TPFCS function calls.

The following describes the ZBROW commands:

Function	Description
ZBROW ALTER	Changes the contents or access mode of a specified collection.
ZBROW CLASS	Displays class name information.
ZBROW COLLECTION	Performs maintenance on a collection.
ZBROW DISPLAY	Displays information about a collection or its contents.
ZBROW KEYPATH	Adds, displays, or removes a key path.
ZBROW NAME	Alters or displays collection name information.
ZBROW PATH	Displays path information for a collection structure.
ZBROW PROPERTY	Alters or displays a property.
ZBROW QUALIFY	Qualifies ZBROW command requests for a data store.
ZBROW RECOUP	Manages recoup indexes.

See *TPF Operations* for more information about the ZBROW commands. See the *TPF C/C++ Language Support User's Guide* for more information about the TPFCS C function calls.

Data Communications

In the early versions of the TPF system, the technology of communication systems forced the TPF system to provide many of the functions that were later provided by a network control program (NCP) running in a communication controller. Subsequently, Internet technology is replacing these older technologies that are often proprietary and require specialized hardware. Internet technology accommodates multiple, diverse underlying hardware technologies by adding both physical connections and a set of conventions. Because of the longevity of the TPF system, it continues to support the older technologies and takes advantage of newer ones.

Non-SNA communications control remains in the TPF system to preserve long-established interfaces and is discussed only minimally; there is an emphasis on SNA communications and Transmission Control Protocol/Internet Protocol (TCP/IP) support.

Non-SNA and SNA Communications: The terminology non-SNA and SNA within the TPF vernacular is rather artificial, being primarily based on whether a message using a particular communications protocol arrives at a central processing complex (CPC) through a communication controller:

- Running network control program (NCP)
- Running emulator program (EP)
- Running 3270 local licensed internal code (LIC).

Messages arriving through an EP controller and 3270 local controller are handled by non-SNA communications control. SNA communications handles the rest. Figure 57 on page 170 shows these various options.

Communications control is a collection of programs that support the various communication protocols that are recognized by the TPF system. Components of communications control manage the communications resources for a TPF computing facility and are an integral part of the TPF control program. This is a fundamental, performance-related difference between the teleprocessing support performed by the TPF system and other teleprocessing access methods (such as Virtual Telecommunications Access Method (VTAM)), used by transaction processing subsystems (such as Information Management System (IMS)), found in a general purpose operating system (such as MVS).

TCP/IP Communications: TCP/IP supports an interconnection of computer networks that provides universal communication services; a computer network is a group of connected nodes used for data communication. A computer network configuration consists of data processing devices, software, and transmission media linked for information exchange.

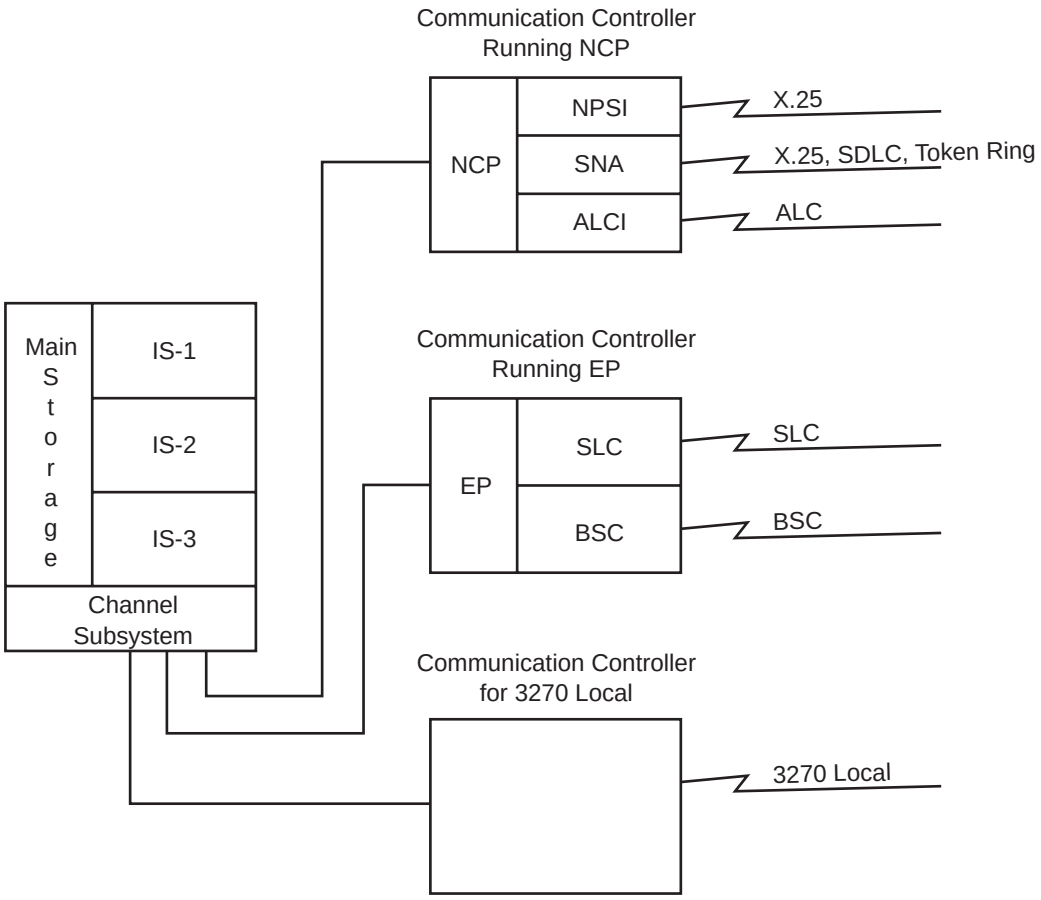


Figure 57. TPF Communication Configuration

Functions of Communications Control

The majority of the work processed by the TPF system originates from terminals, workstations, or other systems, all of which are connected through communication networks.

The major functions of communications control within the TPF system are:

- Handling of different communications protocols
- Identification of *message* source
- Formatting of messages (input and output) based on requirements imposed by application type and terminal or workstation type
- Controlling the flow of message traffic between a central processing complex (CPC) and communication facilities based on the volume of messages
- Error recovery.

Message Routing Overview

The basic tasks of communications control are the gathering of input messages from various sources (the input message is the primary unit of work in the TPF system) and the scheduling of that work according to its priority with respect to other tasks already in progress (see “CPU Loop (Dispatching Work)” on page 61).

Figure 58 on page 174 shows the various steps of communications control that enable the message processing flow within the TPF system. In the following sections, the various components are described in greater detail to give you insight into message processing.

Evolution of Communications Control

Before tracking the path of a typical message through the TPF system, some history is helpful to define the terminology.

One Application and a Simplistic Network

In the 1960s, the TPF system was designed for the airline reservation application, the only application in a TPF system. Therefore, by default, all terminals in a network attached to a TPF processing facility were *logged onto* the only application (the airline reservation application). Also, at that time, the technology of terminal networks was rather simplistic, with communication controllers providing little more function than line multiplexing. As such, the TPF system (also called a host system) was responsible for:

- Requesting input from the terminal interchanges in the network, also called polling.

  In a network where communication controllers provide only minimal function, the host system (the TPF system in this case) must check whether data has been input from terminals. Contrast this with a modern communication controller that performs this function on behalf of the host to which it is attached.
- Scheduling output (that is, responses) to those same terminals.

  In a network where communication controllers provide only minimal function, if a device is not ready to receive data, the data (that is, a response) must be queued by the host system (that is, the TPF system) until the device is ready to receive it. Compare this to a modern communication controller that takes the data from the host and buffers it until the device is ready to receive it.

An application program addressed terminals by physical addresses. The TPF system handled the unique interfaces that were required for each different device type (similar to a communication protocol). The application recognized a terminal by its terminal address in the form of line number (LN), interchange address (IA), and terminal address (TA) or LNIATA.

Multiple Applications and Multiple Destinations

With the advent of the message router in the TPF system, the concepts of multiple applications and message destination were introduced. The concept of destination is the idea that an application can send a message (data) to a terminal or another application. Prior to this, it was assumed that only a terminal could send a message to an application. Also, remember that prior to this, the only application was RES0. The concept of multiple applications introduced the ability to run other applications such as fare quotation and cargo in the airlines industry on the same TPF system. This marked a turning point in the evolution of the TPF system: the separation of system functions from the application functions enabled the TPF system to perform more like an operating system.

To manage the multiple applications and message destinations, the routing control parameter list (RCPL) was invented to carry the origin and destination of an input message throughout the processing of the message in the TPF system. Valid combinations of origins and destinations are:

Origin	Destination
terminal	application
application	application
application	terminal

Another change in communications control at this time was the way in which terminals were identified. Instead of a very specific identification (that is, LNIATA), which is closely related to hardware addressing, terminals were given a symbolic address called a logical end-point identifier (LEID).

In this same era, the log processor was developed because a terminal could *log onto* one of several applications. The log processor controls which application a terminal logs onto.

SNA and Larger Networks

In the 1970s, system network architecture (SNA) support was added to communications control in the TPF system. This allowed the TPF system to be attached to much larger networks that controlled thousands of terminals and workstations.

To maintain the performance factor in the TPF system, use of SNA allowed much of the communications protocol handling to be off-loaded from communications control in the TPF system and to be performed by communication controllers running an NCP. This meant that the TPF system was not responsible for polling input or scheduling output as in the earlier simplistic networks.

In keeping with the trend started with multiple applications and multiple destinations to identify devices symbolically, the resources in an SNA network are called logical units (LU). Each logical unit has an 8-character LU name and a network address. Both an LU name and its network address map into a resource identifier (RID). The RIDs associated with the origin and destination of an input message are carried in the RCPL of the message throughout its processing.

Old and New Applications

Using this history lesson, the terms *old* application and *new* application can now be defined.

The applications that recognize the origin of an input message in LEID form are called *old* applications. And, by default, those applications that recognize an RID as well as an LEID are called *new* applications.

Summary

There are two forms of symbolic identification for resources in a network:

- In a non-SNA network, resources (that is, terminals and workstations) are identified by a logical end-point identifier (LEID).
- In an SNA network, resources (known as logical units (LUs)), are identified by a resource identifier (RID).

  Optionally, SNA terminals can also have LEIDs; an LEID allows an SNA terminal to access old applications, in which case the TPF system transforms the input RID to an LEID, and on output, the LEID is transformed to an RID.

A Communications Overview of Message Processing

The following sections describe the processing that occurs within the TPF system at each stage of the process, from the acceptance of an input message, to the delivery of the output response, to communication facilities.

Figure 58 on page 174 provides an overview of communications control for both SNA and non-SNA networks and shows the processing paths both for a *normal* input message and a *transaction* message. Both of these types of messages arrive at the TPF system in the same way. However, once within the TPF system, the processing paths of the messages diverge until responses are generated; then, the processing paths for delivery of the responses for both types of messages come together again. The processing path of a *normal* input message is described in this section while the processing path of a *transaction* message is described in “TPF Advanced Program-to-Program Communications (TPF/APPC)” on page 183.

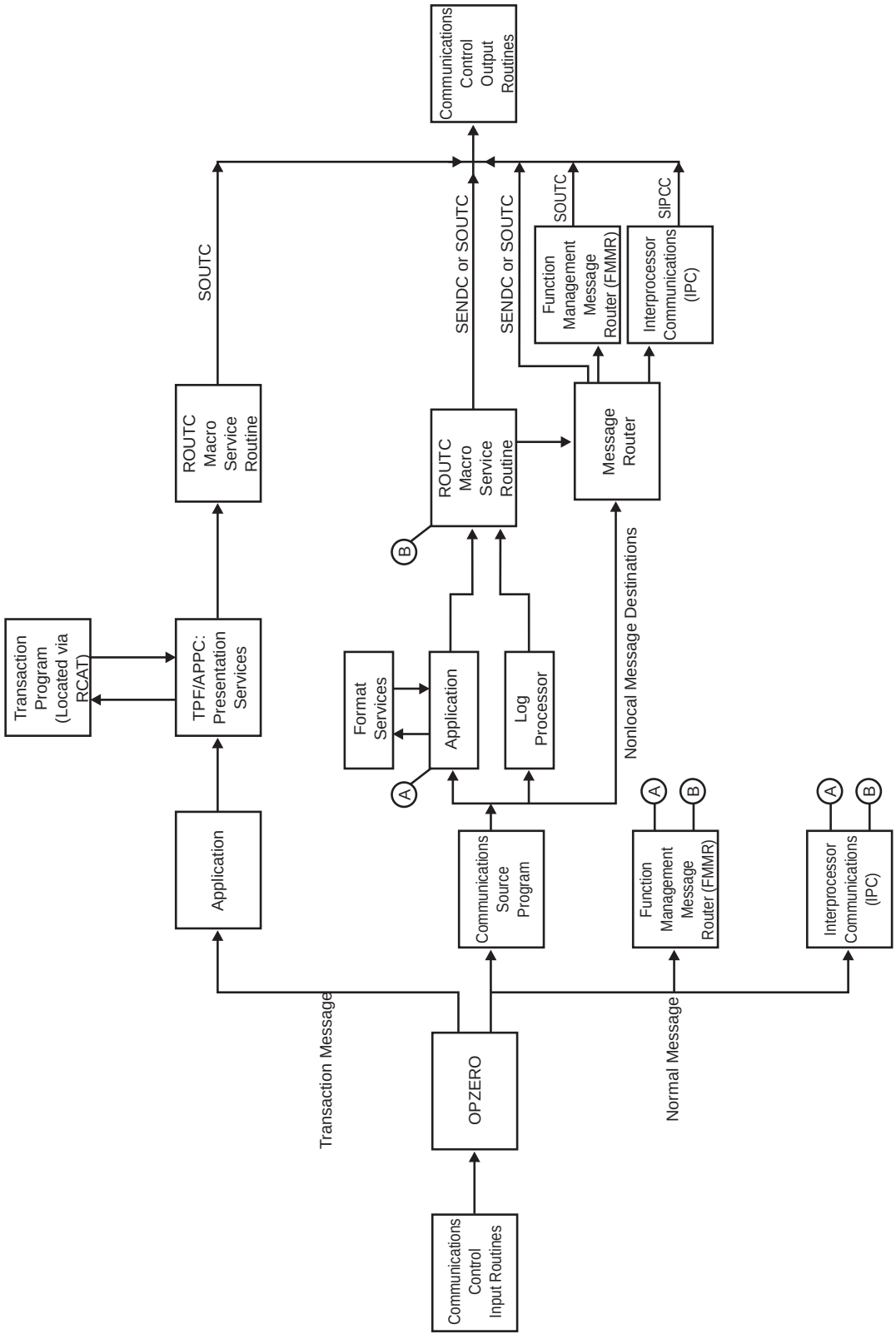


Figure 58. Communications Control Overview

Input Processing

This section describes how an input message arrives at the TPF system through the interrupt mechanism of an I/O interrupt. The I/O interrupt handler passes control to the I/O interrupt handler associated with the communication protocol, which then schedules the input message for processing by placing a work item (associated with the input message) on the input list, a CPU loop list. (The term work item is used here because the exact format of what is put on the input list is dependent on the protocol.)

When the work item gets to the top of the input list (as a result of the other items on the list being removed for processing), control is given to the OPZERO program associated with the communication protocol.

The processing for all sources of messages is described here because this is where the most noticeable differences between SNA and non-SNA are seen.

SNA

- The NCP, executing in a communication controller, polls the communication lines assigned to it, receives data from the lines, and stores the data until the TPF system requests input data.
- When the NCP has data to send to the TPF system, it notifies the TPF system by causing an attention interrupt (a type of I/O interrupt).
- SNA communications control handles the I/O interrupt by ***remembering*** that the interrupt occurred.
- Control is then returned to the program that was executing when the interrupt occurred (see “Significance of Interrupt Handling by the TPF System” on page 35).
- On a periodic basis, the CPU loop in the main I-stream engine polls the communications networks. This causes SNA communications control to request input data (that is, to issue I/O commands) from each NCP that caused an attention interrupt during the preceding time period. Control is then returned to the CPU loop.
- When the I/O from an NCP completes, an I/O interrupt occurs and data that consists of one or more input messages is received in an *input buffer*.
- The I/O interrupt handler places the input buffer on the bottom of the input list.
- When the input buffer (which represents a work item) gets to the top of the input list, OPZERO is given control by the CPU loop.

Non-SNA — Binary Synchronous Communication (BSC): In a BSC network, the TPF system can play several roles:

- For a multipoint link, the TPF system is the control station. This means that because neither the EP nor the BSC network ***tells*** the TPF system when there is data, the TPF system must poll each station in the BSC network to ***ask*** if it has data to send. The BSC poll routine continues to poll until:
  - An input message is sent by a BSC station

    or
  - There is a response (to a previously received input message) from the TPF system waiting to be sent to a BSC station.
- For a point-to-point line, two BSC stations are involved, and the stations must bid for use of the line when there is data to send. (The TPF system is considered a BSC station in this mode.) When a station has data to send, it informs the receiving station, which issues the I/O command to read the data.

Although the concepts of BSC input can be stated simply, the processing required by the TPF system in the management of BSC links is very complex; therefore, its detail is omitted from this discussion.

Non-SNA — Synchronous Link Communication (SLC): SLC is unique in that the TPF system is always ready to accept data from an SLC line. Although you might think that the TPF system is always ready to accept data from other protocols, the difference is that for SLC, the TPF system is constantly issuing I/O commands to read data, whereas for the other protocols, the TPF system only issues I/O commands to read data when it knows there is data to be read.

Suffice it to say, the process required to receive data from SLC is much more complex than the corresponding process to receive data from an SNA network. This is because the TPF system must manage the link, whereas for SNA, similar functions are performed by the NCP.

Non-SNA — 3270 Local:

- When an attention interrupt (a type of I/O interrupt) from a communication controller for 3270 local occurs, the I/O interrupt handler passes control to the local 3270 read routine, which issues the I/O commands to read data from the 3270 local device. Control then returns to the program that was interrupted.
- When the I/O to the device (which proceeds asynchronously) completes, an I/O interrupt occurs that causes the data from the device to be put into a working storage block and a work item associated with the data to be placed on the (bottom of the) input list.
- When the work item gets to the top of the input list, OPZERO receives control.

Operation Zero Program (OPZERO)

OPZERO refers to a collection of system programs associated with communications control in the TPF system. Essentially, there is one OPZERO program for each type of communication protocol.

OPZERO executes only in the main I-stream engine, which is the I-stream engine accepting interrupts associated with all communication facilities except MPIF.

OPZERO Functions for All Protocols: The primary functions of OPZERO are:

- Create an entry control block (ECB)
- Associate the input message with the ECB (the address of the input message is saved in the ECB)
- Pass control to communications source program (COMM SOURCE) to continue input message processing.

OPZERO is activated when the CPU loop removes a work item that is associated with an input message from the input list.

SNA OPZERO: For the BSC, SLC, and 3270 local protocols, the work item is associated with only one input message. However, for the SNA protocol, the work item is associated with an input buffer that contains one or more input messages. Therefore, SNA OPZERO must *deblock* the input buffer, message by message. For each message, SNA OPZERO:

- Obtains a working storage block and moves the first message from the input buffer into the block
- ***Promotes*** the remaining messages in the input buffer and returns that buffer to the top of the input list

The buffer remains at the top of the input list until the buffer is exhausted. This ensures that messages are processed in the order they are received, a requirement for communications control.

SNA OPZERO verifies that the proper protocol has been followed by the originating source. If an input message fails protocol verification, a reply, called a negative response, is generated to indicate the reason the message is rejected.

COMM SOURCE

COMM SOURCE refers to a collection of system programs that transform input messages from their individual protocol-dependent formats into the common system format that is recognized by applications. This relieves the application from knowing the details of the various communication protocols supported by the TPF system.

COMM SOURCE executes only in the main I-stream engine, which is the I-stream engine accepting interrupts from the communication facilities.

COMM SOURCE for All Protocols: The primary functions of COMM SOURCE are:

- Transform the input message from a protocol-dependent format into the common message format called AMSG, which is recognized by both old and new applications
- Determine the destination of the input message; that is, which application is to process the input message

  See “Determining the Destination of the Input Message” for additional information.
- Create a routing control parameter list (RCPL) for the input message

  The RCPL contains information such as the origin of the input message and the destination to route the input message (that is, the name of an application or terminal).
- Collect statistics about the input message for measurement and tuning purposes

  This information includes a count of input messages and a running total of the lengths of each message.
- Allow user exit processing

  User exit processing allows user installation unique processing to be performed.

  See “User Exit Processing” on page 179 for additional information.
- Schedule the input message for processing by an application.

  COMM SOURCE uses system tables to determine the application program segment to process the input message.

  See “Schedule Input Message for Processing by the Application” on page 179 for additional information.

Determining the Destination of the Input Message: Non-SNA terminals are *dedicated* to the TPF system. This means that all input flows directly to the TPF system. However, non-SNA terminals can either select (through a special input message called LOGON) the application in the TPF system they are to be connected with or be *permanently logged* into one specific application. The application is the destination of the message.

SNA devices are *shared* resources, which means that SNA devices are associated with more than one host. A LOGON input message specifies the host and application to be connected (that is, associated) with an SNA device. If an SNA

device is always associated with the same application in a host, it can be *permanently logged*. Again, the application is the destination of the message.

Figure 59 on page 179 shows the tables used to locate the application package necessary to process an input message. A routing control parameter list (RCPL) is constructed in the ECB with fields to hold the origin and destination of the message and the characteristics of the message.

There are several terminal identification tables in the TPF system:

- For a terminal or workstation on an SNA network,
  - The resource vector table (RVT) if the destination application is a *new* application
  - The terminal address table (WGTA) if the destination application is an *old* application
- For a terminal or workstation on a non-SNA network, the terminal address table (WGTA).

A terminal identification table is indexed (accessed) based on the origin of the input message, and the origin is dependent on the communication protocol involved. The tables contain a pointer for each user terminal and workstation and are used to locate the application that the user selected previously.

A program called the log processor is used to interpret a logon input message, which permits a terminal or workstation user (who is the source of input) to select an application package, such as the Airlines Reservation Application Package (). In this publication, *application packages* refers to installation-produced software. The log processor sets up the pointer in the appropriate terminal identification table (RVT or WGTA).

The pointer in the appropriate terminal identification table locates an item in the application name table (ANT), which has a field to hold an application name (such as RES0) to which the user is connected, and another field to point to an item in the routing control application table (RCAT). The RCAT item holds the online linkage to an application program segment. A large application, such as RES0, consists of many program segments, (ranging from hundreds to thousands), only one of which is identified in the RCAT.

The program segment located through the RCAT represents the departure from TPF system programs into installation-written programs. Keep in mind, however, that COMM SOURCE runs in the ECB-controlled environment as does the installation-written program, invoked by COMM SOURCE. For some applications, in particular RES0, terminals and workstations are permanently logged on to a single application. A permanent logon is, in essence, done during system generation.

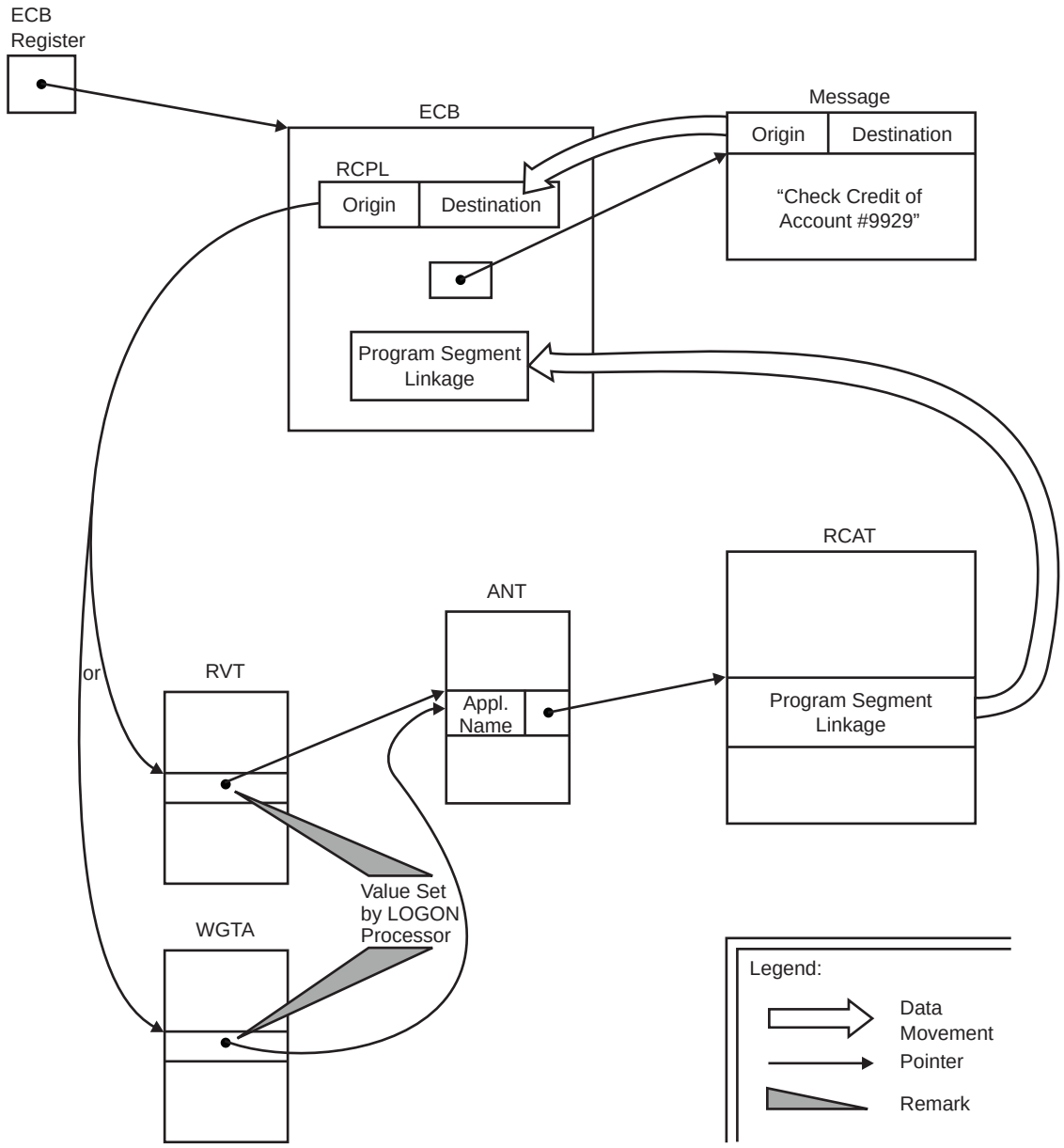


Figure 59. Tables Used to Locate an Application

User Exit Processing: The SNA communications control user exit can be entered to permit user-installation unique processing to be done, such as:

- Editing and changing the text of the input message
- Changing the origin or destination of the input message
- Terminating message processing by discarding the message or rejecting the message with an error message
- Selecting the I-stream engine that is to process the message.

The user exit returns control to COMM SOURCE for additional processing.

Schedule Input Message for Processing by the Application: In a central processing complex (CPC) with multiple I-stream engines, the remainder of the ECB-controlled process can be dispatched to one of several I-stream engines. If the user exit specified the I-stream engine in which to execute the application, a work

item for the ECB is placed on the cross list of this I-stream engine. Otherwise, there is a load balancing routine in the TPF system that places the work item on the cross list of the I-stream engine that is least busy.

The appropriate application program segment on an application I-stream engine is invoked as a result of COMM SOURCE (executing in the main I-stream engine), placing an item on the cross list of an application I-stream engine. Pointing to an ECB of a particular I-stream engine is the essence of moving an Entry between two I-stream engines: the ECB itself remains in the same main storage location while the processing of the Entry is assumed by another I-stream engine in the CPC.

Application Services

Some facilities provided by the TPF system for application use are:

- 3270 simulation

  The purpose for providing the 3270 simulation facility is to shield applications from device dependencies. In particular, 3270 simulation provides transformation of:

  – An 8-bit character code data stream into a 6-bit character code data stream
  – A 6-bit character code data stream into an 8-bit character code data stream.

  Note that 6-bit character codes are a remnant of the early days of the TPF system and that the applications written for 6-bit character codes can be used with 3270 simulation services for processing 8-bit character code data streams.

  Typically, an application input message editor and its corresponding output message sender intercept messages directed between the TPF system and the 3270 to request simulation services.

- 3270 mapping

  The purpose for providing the 3270 mapping facility is to shield application design from being device dependent. Secondly, the application program does not need modification when the end user of the application requires changes to an input message format, display format, or print listing.

  Maps, which are designed by a user installation for a particular application, specify field definitions and constant text (such as titles in a display or column headings in a print listing) for input and output displays and print listings (output only) that are associated with the application.

  An application requests input mapping services to remove the constant text and device-dependent control characters from an input message. The resulting input message consists of only the data that is necessary for processing. An input map is used to control the removal of the constant text.

  Output mapping is used when an application formats an output message that only contains the variable data in an output display or print listing. The application requests output mapping services to add the constant text and device-dependent control characters to the output message. The resulting output message is then in the proper format for the receiving device.

  Despite its name, 3270 mapping is applicable to devices other than 3270-type devices, such as 1980-24 and 1977 devices.

- Terminal control block

  The applications invoked by COMM SOURCE are frequently installation-written *transaction processing editors* that select one of the many message processing programs. If the processing of the current message is dependent on previous message processing, the transaction processing editor can retrieve a terminal control block. The terminal control block is used to accumulate data from the

many messages that make up a transaction. (Single message transactions do not require the retrieval of a terminal control block.)

The TPF system supports several such blocks:

- Scratchpad area (SPA) — for use by *new* applications
- Routing control block (RCB) — for use by *old* applications
- Agent assembly area (AAA) — for use by airline reservation applications, such as RES0
- Node control block (NCB) — for use by SNA communications control to assemble an input message or to queue output messages.

A terminal control block is obtained by using the origin address field of the RCPL that is associated with the input message as the basis of an index into a file holding these blocks.

- The application program interface (API) to send an output message (that is, a response to an input message) using the output message router program to a terminal or workstation in a network
  - The output message that an application formats is built in a working storage block. If the output message cannot be contained in a single working storage block, the rest of it is built in pool records located on file storage.
  - The application program formats a routing control parameter list (RCPL), which specifies information such as the origin and destination of the output message
  - The application program issues the ROUTC macro, which invokes the output message router.

Output Message Router

The output message router is the service routine for the ROUTC macro. Based on the destination specified in the RCPL, the message characteristics (one working storage block with or without pool records), and network activity, the output message router transforms the message into a device-dependent format and transmits the message over the network.

Message Destination: The type of destination is:

- A terminal indicated by an RID or LEID in the RCPL

or

- An application indicated by a sequence of alphanumeric characters in the RCPL.

The destination of the message is associated with a CPC, which is either local relative to the CPC that is transmitting the message, or remote. *Local* implies that the destination CPC is the same as the originating CPC. If the destination CPC is remote, the destination CPC is either in complex or out of complex.

In complex implies that the complex is loosely coupled and that the destination CPC is within the loosely coupled complex. Out of complex implies that the destination CPC is not one of the CPCs in the loosely coupled complex of the CPC that is originating the message.

In Figure 60 on page 183, if the originating CPC is A, and

- The destination CPC is A, the destination is local
- The destination CPC is B, the destination is remote and in complex
- The destination CPC is C, the destination is remote and out of complex.

Message Buffering: In order to transmit a message, it must be sent in increments that are acceptable to the device. Communications control performs the buffering of messages based on the characteristics of the device.

Message Queuing: To regulate the rate of message flow from the TPF system to prevent an overload of traffic on the network, messages are queued on DASD (in pool records), if necessary, until traffic permits transmission. This is called pacing.

Message Recovery: When a network is defined, message recovery can be specified for a range of resources. This means that there must be the capability to retransmit any message sent by or received by the resource.

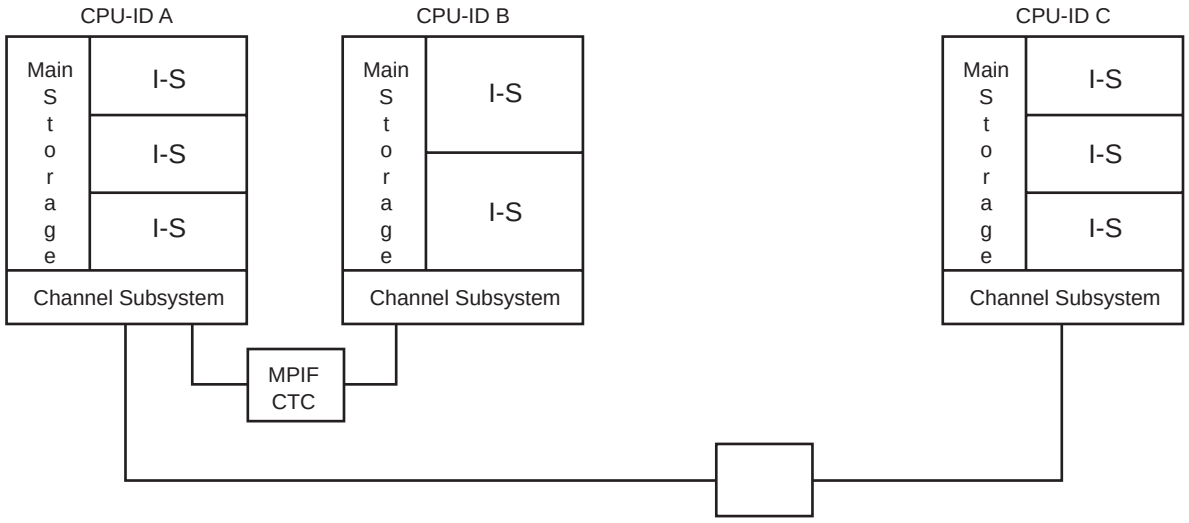
In the TPF system, by definition, when the input message is recoverable, the output message is recoverable. This is accomplished by saving a copy of the message on file storage (in a pool record) until positive indication is received by the TPF system that the message has reached its destination.

Transmission of Output Message: A message transmitted over an SNA network is sent from the TPF system using the SOUTC macro. A message for the non-SNA network is sent from the TPF system using the SENDC macro.

Functional Management Message Router (FMMR): FMMR processing occurs when the output message router detects, through the content of an RCPL destination routing, that a message is to be sent to an application in a CPC in a TPF system whose CPU ID is external (that is, remote and out of complex) to the originating complex. The output message router is invoked to do this either through a ROUTC macro request or as a result of an enter from COMM SOURCE.

In effect, COMM SOURCE serves as an intermediate *routing node*.

Interprocessor Communication (IPC): IPC processing occurs when the output message router detects, through the content of an RCPL destination routing, that a message is to be sent to another CPC in the LC complex (that is, remote and in complex).



Origin CPU-ID	Destination CPU-ID	
A	A	Local
A	B	Remote, In Complex
A	C	Remote, Out of Complex

Figure 60. Destination — Output Message Routing

TPF Advanced Program-to-Program Communications (TPF/APPC)

TPF Advanced Program-to-Program Communications (TPF/APPC), an implementation of IBM's Advanced Program-to-Program Communications (APPC), is an interface that allows TPF transaction programs to communicate with nodes that have implemented APPC.

The alternate processing path out of OPZERO shown in Figure 58 on page 174 pertains to the processing of a transaction message. A transaction message is an input message that originates from an LU 6.2 type SNA resource. A transaction message is processed by a special type of application program called a transaction program.

A conversation takes place between a transaction program in the TPF system and a transaction program residing in another LU 6.2 resource. That is, the transaction programs take turns processing the information received from each other.

When COMM SOURCE recognizes a transaction message, it passes the message to the input function of TPF/APPC presentation services, which uses the transaction program name table (TPNT) to determine the specific transaction program to perform message processing for a new conversation. For an established conversation, the transaction program handles the message. On the output side, the output functions of TPF/APPC presentation services use ROUTC macro services.

TPF MQSeries Support

The IBM MQSeries product is one of the fastest-growing, message-oriented middleware products in the industry. It has the ability to deliver messages to a diverse heterogeneous set of systems with a single set of application programming interfaces (APIs).

The TPF system supports an MQSeries local queue manager, MQSeries client, and an MQSeries server. If the application connects (using the MQCONN API) to a queue manager other than the local queue manager, all API requests will be sent over TCP/IP or LU 6.2 connections to the remote MQSeries server. The complete functionality of the remote MQSeries server is available to the application. If the application connects to the local queue manager, the TPF MQSeries local queue manager will then service subsequent API requests. For more information about TPF MQSeries support, see *TPF Application Programming*.

Local Queue Manager

The implementation of TPF MQSeries local queue manager support is based on a subset of the standard MQSeries local queue manager support interface used by other MQSeries product offerings. The following is a list of the standard MQSeries functions that are supported by TPF MQSeries local queue manager support:

- An ISO-C interface is provided for the following supported functions that make up the Message Queue Interface (MQI):
  - MQBACK
  - MQCLOSE
  - MQCMIT
  - MQCONN
  - MQDISC
  - MQGET
  - MQINQ
  - MQOPEN
  - MQPUT
  - MQPUT1
  - MQSET.
- For TPF MQSeries local queue manager support, ZMQSC commands allow you to do the following:
  - Change and define channels, processes, profiles, and queues
  - Delete or display channels, processes, and queues
  - Reset, resolve, start, stop, or trace channels
  - Display, start, or stop the queue manager
  - Move messages from one transmission queue to another.
- The TPF MQSeries local queue manager supports three queue types:
  - Remote
  - Local
  - Alias.
- *Remote queues* belong to a remote queue manager.
- *Local queues* can be normal or transmission. Local queues can be normal queues or transmission queues. Normal local queues can be processor unique or processor shared. Transmission queues are always processor unique. Processor

unique queues reside in memory and are made persistent by filing the memory copy of the queue to fixed file TPF records on a regular basis (this is called checkpointing) and logging any updates between checkpoints to the TPF recovery log. Processor shared queues reside on file using TPF collection support (TPFCS).

- *Alias queues* are queues that are named as an alias for a queue defined to the local queue manager.

A TPF MQSeries message channel agent (MCA) is provided to communicate with adjacent MQSeries systems by using TPF sender, receiver, and server connection channels only. Channels communicate by using Transmission Control Protocol/Internet Protocol (TCP/IP) exclusively; therefore, a TCP/IP connection between the TPF 4.1 system and a remote MQSeries system is required.

- The following channel user exits are provided:
  - The TPF MQSeries assign LNIATA user exit, CUIW, and the TPF MQSeries convert to object handle user exit, CUIV, are ROUTC bridge user exits that are provided to help customers route messages originating from the MQSeries queue manager to non-MQSeries TPF applications, and then return the message to the MQSeries network.
  - The TPF MQSeries channel message user exit in segment CUIT allows you to process a channel message.
  - The TPF MQSeries channel message retry user exit in segment CUIT allows you to try to put a message to a destination queue if a previous attempt failed.
  - The TPF MQSeries channel message security user exit in segment CUIT provides security for data that is sent or received over TPF MQSeries channels.
  - The TPF MQSeries queue trigger user exit, CUIR, is called the first time a message arrives on the queue if no process object is associated with the queue.

System administrators have the following network routing options when defining remote queues:

- *Local definition of remote queues* a remote queue can be defined as a local definition of a remote queue allowing the administrator to determine the destination queue manager and destination queue rather than the application.
- *Queue manager aliasing* a remote queue can be defined with a queue manager alias where the administrator determines the destination queue manager rather than the application.

TPF MQSeries supports the following message types:

- *Nonpersistent messages* are not guaranteed to be delivered and will not survive an initial program load (IPL) of the TPF 4.1 system. Inbound nonpersistent messages over a fast channel are delivered immediately to the TPF-unique MQSeries ROUTC bridge.
- *Persistent messages* are guaranteed to be delivered and will survive an IPL of the TPF 4.1 system.

TPF MQSeries supports the following channel types:

- *Sender channels*: Normal sender channels transmit all messages in batches that are guaranteed to be delivered. Fast sender channels do not guarantee delivery of nonpersistent messages.
- *Receiver channels*: Normal receiver channels receive persistent and nonpersistent messages and guarantee delivery of all messages. Fast receiver

channels receive both persistent and nonpersistent messages. Nonpersistent messages that are received over a fast channel are passed immediately to applications through the TPF MQSeries ROUTC bridge.

- *Server connection channels* are defined on the server running the queue manager to communicate with an MQSeries application running in an MQSeries client environment.

The following unique TPF functions are provided with TPF MQSeries local queue manager support:

- *TPF MQSeries ROUTC bridge* immediately passes nonpersistent messages that arrive at TPF MQSeries fast receiver channels to applications. Nonpersistent messages that arrive through the MQSeries interface are routed to a non-MQSeries TPF application.
- *Swing queue* provides the ability to move messages from one transmission queue to another because of the high volume of messages that TPF applications process and to prevent a continuous buildup of messages on a transmission queue whose channel is down.
- *Movemsgs* allows you use the ZMQSC MOVEMSGS command to move messages in a memory queue from a deactivated processor to another processor in a loosely coupled complex.

Communication Interfaces

The TPF system supports a variety of network protocols with most being supported through the systems network architecture (SNA). These include SDLC, CTC, X.25, ALC, and Token Ring. The non-SNA protocols are BSC, SLC, and 3270 local.

The primary SNA interface is through channel-connected communication controllers running NCPs, although a CTC interface is also supported. The TPF system recognizes a wide variety of SNA LU types (LU0, LU1, LU2, LU3, LU 6.2) with a wide variety of terminal types (such as 3600/4700, 327x, 328x, PS/2, AS/400, RISC System/6000).

In a loosely coupled complex, emulator program (EP) protocols (that is BSC and SLC) must be connected to a single CPC, known as the EP processor.

As a participant in a full function network, the TPF system supports two different interfaces to the SNA network:

- Subarea interface (T5)
- Low entry network (LEN) interface (T2.1).

The function of the TPF system within an SNA network is to act as a data host where network information and management are assumed to be handled by a VTAM communications management configuration (CMC). The TPF system manages the applications within its complex, the channel interfaces to locally attached communication controllers, and the range of terminals and LU types connected to the TPF system.

Error Recovery

In keeping with the TPF philosophy to recover quickly when a system error occurs, there are several design points in communications control for this purpose:

- In an SNA network, the TPF system reduces the number of network restarts due to a system failure.

A network restart requires a considerable amount of time if the SNA network is large.

The TPF system checkpoints network and session status periodically. This means that the system tables held in main storage are written to file storage. Therefore, in the event of a TPF system failure, the system tables can be recovered as of the last checkpoint time.

Practice has shown that most of the time, the network is not aware of gaps in the operation of the TPF system.

In addition, during a system restart, the TPF system queries the network to find out current network and session status. The system tables are updated with this information and checkpointed (that is, written to file storage). Thus the TPF system dynamically attempts to maintain the latest information about the network and its sessions.

- When an anomaly is detected, the TPF system attempts a recovery. Recovery includes re-synchronization of sessions. This process is especially useful after an interruption in TPF system operation. However, it is also used during normal operation to recover from other types of errors.

  Network status and timers are used to detect deadlocks that can occur because of the unpredictable rate of message traffic.

Function Management Message Router (FMMR)

Functional management message router (FMMR) is a TPF routing mechanism that permits an application in a TPF system to send a message (data) to an application in another TPF system. That is, the destination TPF system is *remote and out of complex* to the origin TPF system. Therefore, if the origin TPF system is a uniprocessor, all other CPCs running the TPF system are remote. If the origin TPF system is loosely coupled, then the destination TPF system is a CPC that is external to the origin loosely coupled complex.

Only the SNA link protocol is supported for sending a message to a remote TPF application.

Interprocessor Communications (IPC)

Interprocessor communication is the mechanism that is used for communication between the CPCs in a loosely coupled (LC) complex; that is, remote and in complex.

Conceptually, IPC can be viewed as a communication facility that is internal to the TPF system and uses the MPIF channel-to-channel protocol.

User Exits

The TPF system provides a variety of user exits to allow a user installation to tailor system processing to address unique requirements. There are user exits within communications control, such as message recovery, transaction routing for input messages, ROUTC for output messages, COMM SOURCE, and processing selection vectors, which are activated based on the LU name of the input source.

Transmission Control Protocol/Internet Protocol (TCP/IP) Support

TCP/IP supports an interconnection of computer networks that provides universal communication services. The TPF system provides:

- TCP/IP offload support
- TCP/IP native stack support.

TCP/IP offload support is an offload implementation based on the TCP/IP offload device. Socket applications in the TPF system communicate through the TCP/IP offload device to applications in remote TCP/IP devices.

The socket application programming interface (API) consists of standard ISO-C socket function calls. With TCP/IP offload support, when a TPF socket application issues a socket function call, the call is sent to the TCP/IP offload device using the Common Link Access to Workstation (CLAW) protocol. The TCP/IP offload device then communicates with the remote node using TCP/IP. The TCP/IP offload device uses the CLAW protocol to pass the socket API return code back to the TPF system, which then presents the return code to the socket application that issued the socket function call.

With TCP/IP native stack support, the stack is incorporated in the TPF system itself. This support enables the TPF system to directly connect to IP router boxes in addition to continuing to support offload devices.

See *TPF Transmission Control Protocol/Internet Protocol* for more information about TCP/IP support.

Internet Daemon

The *Internet daemon* is a socket application that creates and monitors sockets with remote nodes on the Internet and then starts Internet server applications that process the specific communications protocol used with the remote nodes. Once a connection is made, the Internet server application communicates with the remote node on the Internet directly.

Internet server applications are also socket applications. For example, a Trivial File Transfer Protocol (TFTP) server, a File Transfer Protocol (FTP) server, and a Hypertext Transfer Protocol (HTTP) server are socket applications.

Figure 61 on page 190 shows the relationship of the Internet daemon and Internet server applications.

The Internet daemon is a long-running process that consists of two major components:

- The Internet daemon monitor, which is responsible for starting and stopping the Internet daemon listeners for Internet server applications and for error recovery when an Internet daemon listener fails
- An Internet daemon listener, which monitors the Internet server applications and, with some process models, creates and monitors a socket for the Internet server application.

There are process models that define the interface to the Internet daemon based on the level of control needed by an Internet server application. There are subtle differences in the functions used by the Internet daemon, which relate to the level of control that results.

The WAIT, NOWAIT, or DAEMON process model provides synchronous control because the `tpf_fork` function is used to create a child process for which the TPF system sends a SIGCHLD signal to the parent process when the child process ends. The DAEMON process model differs from the WAIT and NOWAIT process models in that the Internet daemon ***does not*** create or monitor sockets.

The AOR process model provides asynchronous control because the `activate_on_receipt` or `activate_on_receipt_with_length` function is used. When a remote client connects, the Internet daemon issues an `activate_on_receipt` or `activate_on_receipt_with_length` function to pass control of the new socket to your TCP server application when the first message is received from the remote client.

The NOLISTEN and RPC process models do not provide any control; the `swisc_create` function is used to create an independent ECB.

Syslog Daemon

The syslog daemon is a server process that is started by the Internet daemon and receives messages on well-known port 514 as shown in Figure 61 on page 190. Internet server applications and components use the syslog daemon for logging purposes and can also send trace information to the syslog daemon. Messages can be logged to files or to tape. See *TPF Transmission Control Protocol/Internet Protocol* for more information about the syslog daemon.

File Transfer Protocol (FTP) Server

An FTP server is a socket application that is called by the Internet daemon when a message is received on well-known port 21. The FTP server is started by the Internet daemon and subsequently communicates with the remote node directly as shown in Figure 61 on page 190.

Trivial File Transfer Protocol (TFTP) Server

A TFTP server is a socket application that is called by the Internet daemon when a message is received on well-known port 69. The TFTP server is started by the Internet daemon and subsequently communicates with the remote node directly as shown in Figure 61 on page 190.

Hypertext Transfer Protocol (HTTP) Server

An HTTP server is a TCP/IP application that is called by the Internet daemon when a message is received on well-known port 80. The HTTP server is started by the Internet daemon and subsequently communicates with the remote node directly as shown in Figure 61 on page 190.

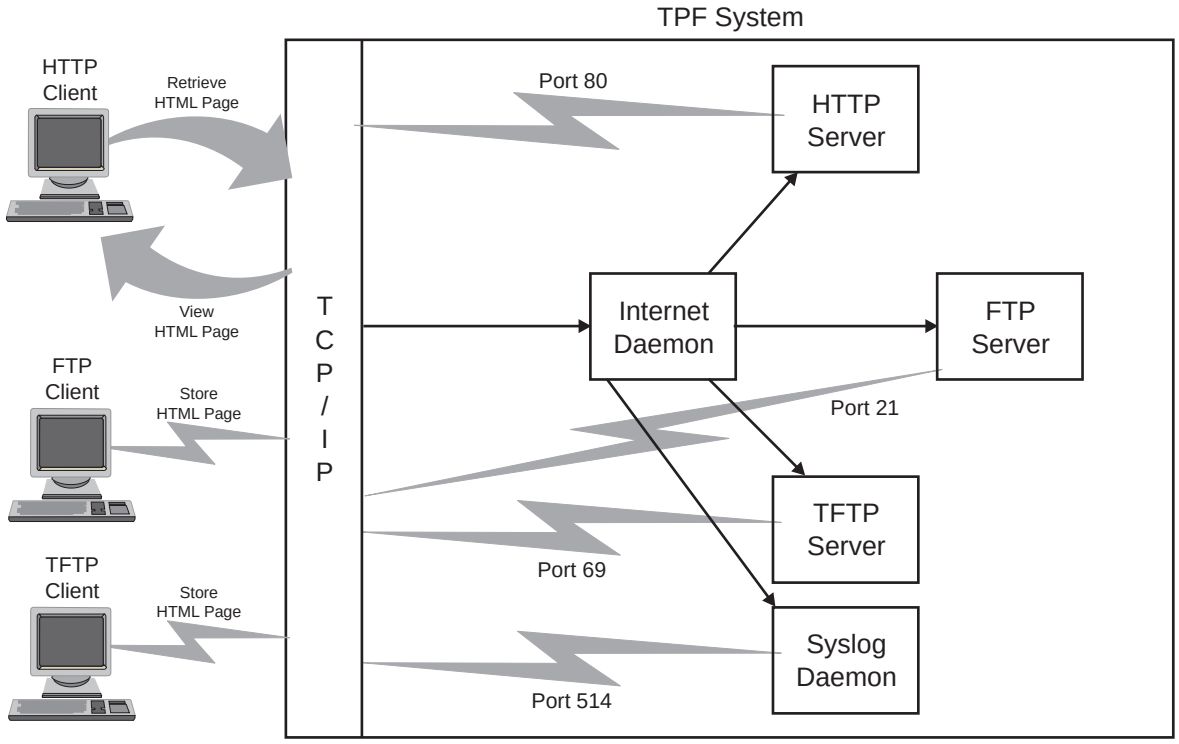


Figure 61. Socket Application Overview. HTTP, FTP, TFTP, and syslog daemon servers communicate over well-known ports.

TPF Internet Mail Server Support

TPF Internet mail server support provides a set of servers that implement the standard Internet mail protocols on the TPF system. Users, or mail clients, interact with the TPF Internet mail servers to send and retrieve Internet mail, also known as electronic mail (e-mail). The TPF system supports the following standard Internet protocols:

- Simple Mail Transfer Protocol (SMTP)
- Internet Message Access Protocol (IMAP) Version 4
- Post Office Protocol (POP) Version 3.

See *TPF Transmission Control Protocol/Internet Protocol* for more information about TPF Internet mail server support.

Remote Procedure Call (RPC) Server

RPC allows applications on one workstation to call functions that reside on and are run by another workstation. A RPC server is a TCP/IP application that is started by the Internet daemon. A RPC server receives messages on a user-defined port. The RPC library application programming interfaces (APIs) establish all required client\server connections by using socket APIs. See *TPF Application Programming* for more information about RPC.

TPF Internet Server Support

TPF Internet server support is based on the Portable Operating System Interface for Computer Environments (POSIX) standards wherever possible.

A person who is surfing the Internet can obtain information from a TPF system as shown in Figure 62 on page 192. Information is in the form of:

- Web pages

  Web pages, stored as stream files in Hypertext Markup Language (HTML) format in the file system, are retrieved by an HTTP server.
- TPF data

  The TPF system provides a front end for starting a TPF application from the Internet and application programming interface (API) functions to send output over the Internet.

The TPF system provides the following functions:

- An Internet daemon that receives Internet requests and starts Internet server applications
- An FTP server for file transfer services, which can be used to store Web page content in the file system.
- A TFTP server for file transfer services, which can be used to store Web page content in the file system.
- A set of servers that implement the standard Internet mail protocols. See *TPF Transmission Control Protocol/Internet Protocol* for more information about TPF Internet mail server support.

The TPF system supports the following types of transport protocols:

- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP).

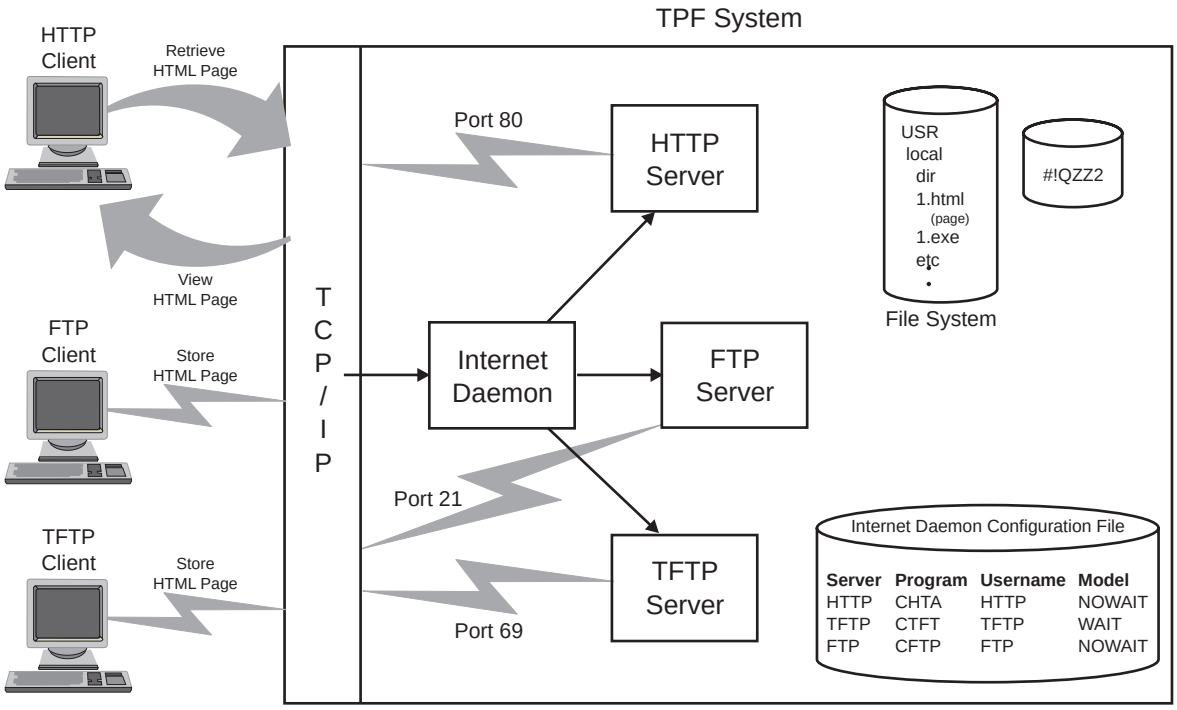


Figure 62. Overview of TPF Internet Server Support. An FTP or TFTP client creates and maintains Web page content using FTP or TFTP write requests. An HTTP client is surfing the Internet through HTTP read requests.

Storing Web Page Content in the TPF System

Web page content is created on another system such as UNIX or a personal computer (PC) and transferred to the TPF system using the FTP or TFTP server as shown by Figure 63 on page 193. Web page contents are stored as stream files in the file system.

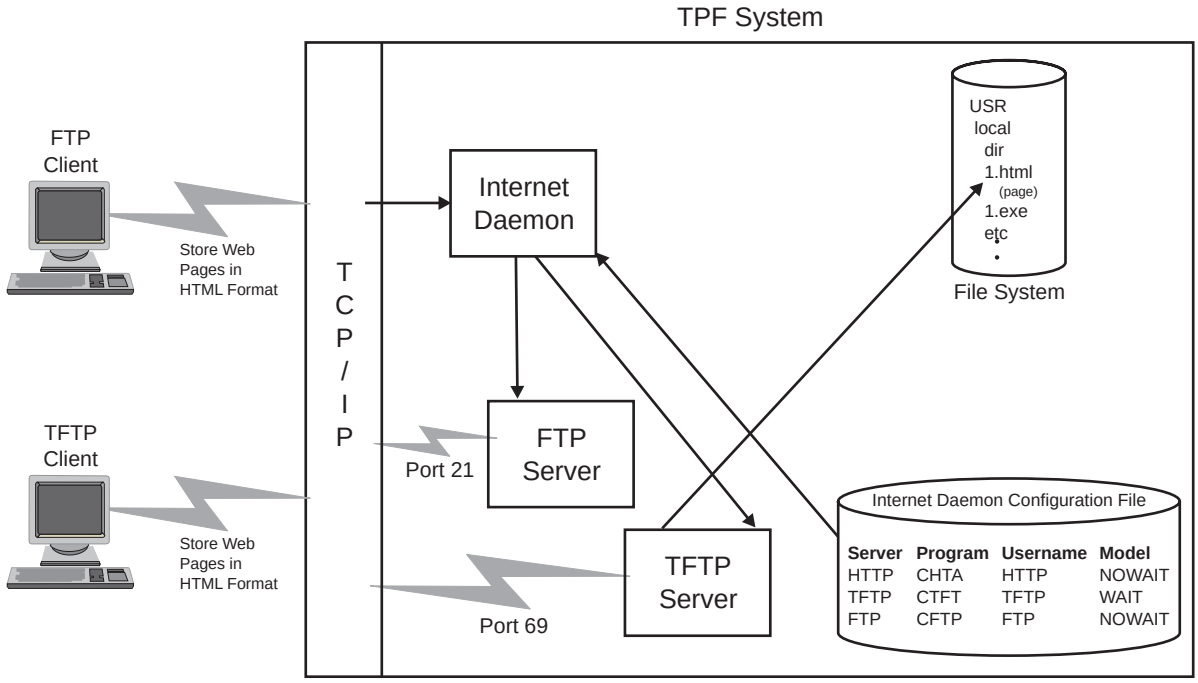


Figure 63. Storing Web Page Content in the TPF System. The FTP or TFTP server stores the 1.html file containing Web page content in the file system as a result of an FTP or TFTP write request.

Retrieving Web Pages from the TPF System

When a request is received over a TCP/IP network, the Internet daemon starts an HTTP server to retrieve the Web page from a file in the file system and to return the Web page content over the Internet as shown in Figure 64 on page 194.

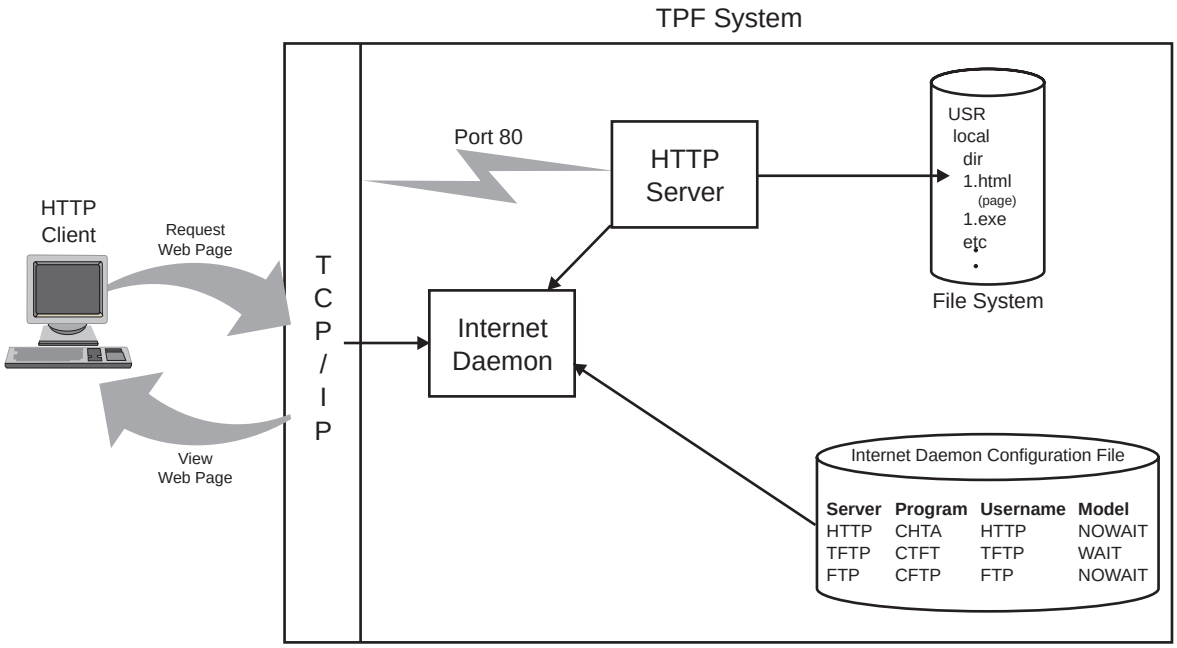


Figure 64. Retrieving Web Pages from the TPF System. An HTTP client requests a Web page from the 1.html file in the file system using an HTTP read request. The Internet daemon starts the HTTP server to retrieve the Web page and send it over the Internet.

Starting a TPF Application from the Internet

An *executable script*, a type of executable file in the file system, can be used to start a TPF application from the Internet. The TPF application can direct its output to the Internet rather than to an agent.

Figure 65 on page 195 shows that the executable script is called 1.exe in the file system and the TPF application is a loader E-type program called QZZ2. When the client requests information, the Internet daemon starts the HTTP server to start program QZZ2 using an executable script, the 1.exe file, to retrieve the requested data and send it over the Internet.

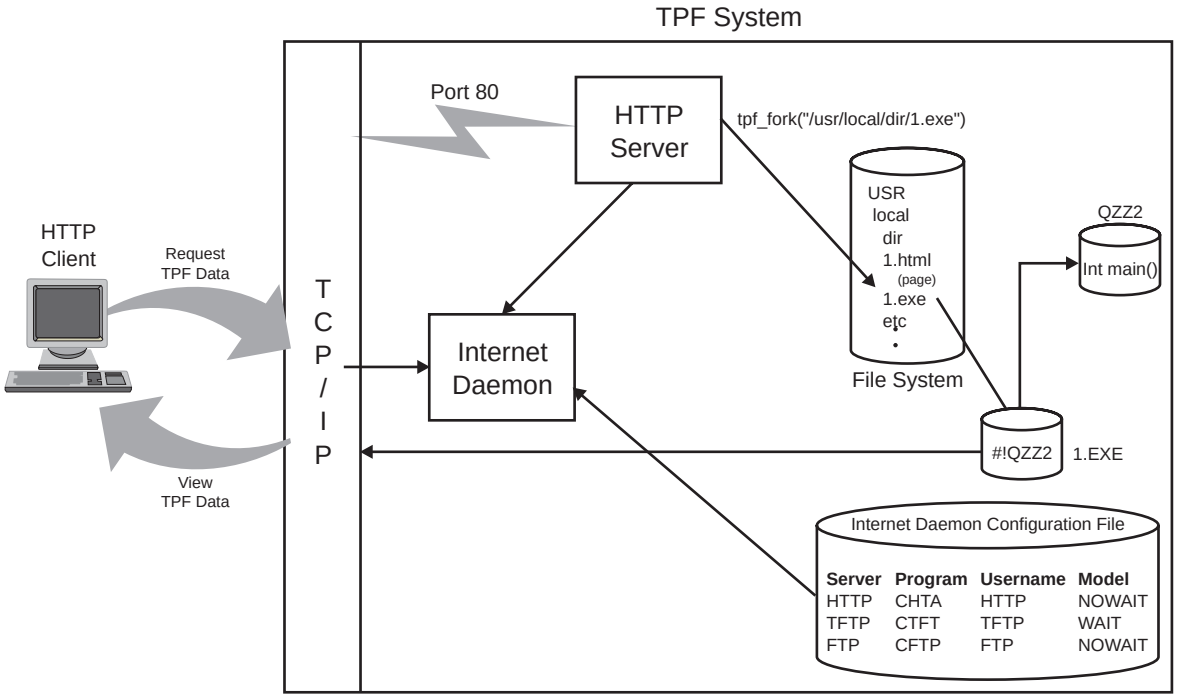


Figure 65. Starting a TPF Application from the Internet. An HTTP client requests TPF data using an HTTP POST request to start the TPF program named in the 1.exe executable script.

Index

Special Characters

$ADPC macro 87
$CRISC macro 87

Numerics

3270 local 169, 176, 186
3270 mapping 180
3270 simulation 180
4K common frame 74
  common block 74
  relationship of common frame to common block 75
4K frame 74
  ECB private area (EPA) 75
  relationship of frame to logical block 74

A

absolute address 26, 27
absolute path name 153
access permissions 150
accessing a collection by key 162
accessing non-standard records
  on a general data set (GDS) 143
accessing standard records
  on a general data set (GDS) 143
add work to a list on specified I-stream macro ($ADPC) 87
address
  absolute address 26
  real address 26
  virtual address 26
address space 26
  home virtual address space 26
  isolation 29
  primary virtual address space 26
agent assembly area (AAA) 180
agents 3
aging out process 139
airlines line control (ALC) 186
airlines reservation application package (RES0) 171, 178
animation 39
animation of a program 108
application enabling tools 149
application I-stream engine 48, 49
application lock 47
application name 178
application name table (ANT) 178
application program 39, 83
  transaction program 183
application programming interfaces (APIs), TPFCS 167
application services
  3270 mapping 180
  3270 simulation 180
  application program interface (API) to send output message 181
application services *(continued)*
  communications control 180
application, new 172
application, old 172
applications, multiple
  destinations, multiple 171
arbitration 29
  hardware storage 29
architecture of the TPF system
  premise 59
archives, database 166
archiving support, TPFCS 166
array collection 163
attributes
  data record 102
  logical device type 102
  record duplication 102, 121
  record longevity 103
  record size 102
authorization
  relative to supervisor state 35

B

backup record 102, 121
bag collection 163
band number 107
basic subsystem (BSS) 130
  operation of system 135
begin transaction 12
benchmark message 13
benefits of TPFCS 159
binary large object (BLOB) collection 163
binary synchronous communication (BSC) 175, 186
BLOB collection 163
block size, standard 74
block type 74
browse support, ZBROW commands 168

C

C functions, using in C programs 151
capture and restore support, TPFCS 166
categories of
  I-stream engine 48
central processing complex (CPC) 1, 37
  control structure 54
  description 37
  interprocessor communications facility (IPC) 182
  loosely coupled multiprocessing 49
central processing unit (CPU) 21, 24
  serialization 31
CF cache support 50
channel subsystem 1, 32, 49
channel to channel (CTC) 186
characteristics, of TPFCS collections 161
circular waiting 23

collection lifetimes 164
collections, TPFCS 161, 163
COMM SOURCE 62, 63, 177
COMM SOURCE (communications source program) 67, 68
  functions 177
  OPZERO passes control to 58
  routing control parameter list (RCPL) 177
  selection of application program 58
  user exit 179
command 91
  contrast with instruction 32
commands, TPFCS 167, 168
commit transaction 12
commit/rollback protocols
  TPF transaction services 166
common block 74
common I/O handler (CIO) 89
communication controller 169
  3270 local 169
  emulator program (EP) 169
  network control program (NCP) 169
communications control 10
  application services 180
  cross list 180
  description 169
  destination of a message 171
  error recovery design 186
  functions 170
  history of 171
  input message 169
  input processing 175
  input processing for non-SNA, 3270 local 176
  input processing for non-SNA, BSC 175
  input processing for non-SNA, SLC 176
  input processing for SNA 175
  load balancing 180
  message destination 171
  message processing overview 173
  multiple applications 171
  network control program (NCP) 175
  network restart 186
  non-SNA communications control 10
  one application 171
  output message router 181
  output message transmission 182
  pacing 182
  relation to performance 169
  restart due to a system failure 186
  sending output message 182
  SNA communications control 10
  SNA networks 172
  TCP/IP 187
  user exit 179, 187
communications management configuration (CMC) 186
compare and swap instruction (CS) 30
compare double and swap instruction (CDS) 30
computer room agent set (CRAS) 91
concurrency controls
  none (nonlocking cursor) 165
concurrency controls *(continued)*
  optimistic 165
  pessimistic 165
concurrency filter lock facility (CFLF) 32
concurrent processes 22, 54
console 91
control bits 106
control diagram 53
  control structure diagram 53
  control transfer diagram 53
control of updating a single record 108
control program 56
  description of 80
  reentrancy 80
  shared among multiple I-stream engines 80
control structure diagram 53
control structure for the TPF system 56
control transfer 83
control transfer diagram 53
conversation 183
converting symbolic file address 105
core block reference word (CBRW) 89
counter, update sequence 165
coupling facility record lock support
  lock maintenance 104
  loosely coupled considerations 139
  overview 50
  parallel processing 50
coupling facility support 50
CPU affinity 47
  advantage 48
  application I-stream engine 48
  disadvantage 48
  main I-stream engine 48
CPU loop 56, 58, 61
  cross list 58
  deferred list 58
  input list 58
  processing 76
  ready list 58
  running in all other I-stream engines 84
  running in the main I-stream engine 84
  summary of processing 58
CPU loop list 61, 76
  add an item to list 76
  control transfer 83
  create new ECB and transfer control macro (CXFRC) 83
  cross list 61, 84
  deferred list 61
  delete an item from list 76
  input list 61
  priority 61
  ready list 61
create a low-priority deferred entry macro (CREXC) 88
create a new ECB 83, 88
create a new ECB and transfer control macro (CXFRC) 83
  mode of operation 83
create a new ECB for immediate entry macro (CREMC) 88

create a new ECB with attached core blocks macro (CREEC) 88
create a new Entry 88
create a time-initiated entry macro (CRETC) 88
create macros 88
  create a low-priority deferred entry macro (CREXC) 88
  create a new ECB for immediate entry macro (CREMC) 88
  create a new ECB with attached core blocks macro (CREEC) 88
  create a time-initiated entry macro (CRETC) 88
  create new synchronous ECBs macro (CRESC) 88
  types 88
create new synchronous ECBs (CRESC) 88
critical code 55
critical region 23, 29, 44
  serially reusable program 42
cross list 61, 84
  structure 85
cross-subsystem access service request macro (CROSC) 136
crossover to another I-stream macro ($CRISC) 87
current directory 152
cursor support 165
Customer Information Control System (CICS) 6
cycle down 139

D

daemon
  Internet 188
  syslog 189
DASD
  hardware maintenance 147
DASD control unit
  external lock facility (XLF) 21, 32
  limited lock facility (LLF) 32
  loosely-coupled complex 144
  module record cache 142
DASD record
  in main storage 137, 138
  in module cache memory 142
DASD record caching
  candidacy characteristics 104
  fast write 143
  record ID attribute table (RIAT) 104
  retentive write 143
DAT-mode bit 26
data
  integrity of critical data 145
  interface between offline and online components 143
  loading data to the TPF system 149
  logically separation 98
  overview of database 95
  physical residence 102
  physically separation 98
  residence of 102
data collection 59
data collection and reduction 59
data definition 164
data event control blocks (DECBs)
  FAC8C macro 105
  format of 71
  usage 160
data host 186
data level 75, 158, 160
data organization 95
  factors affecting performance 95
data record
  backup record 102
  duplication 102, 121
  longevity 103
  primary record 102
  size 102
data record attributes 102
data record size 102
  1055 102
  381 102
  4095 102
  4K 102
  large 102
  non-standard on a general data set (GDS) 143
  small 102
data reduction 59
data store
  definition of 161
  initializing 167
database
  design of organization 112
  example of 95
  expansion capability 129
  increase capacity of 102
  index 97
  integrity 104, 145
  loosely coupled multiprocessing 144
  organization design 112
  tightly coupled multiprocessing 144
  utilities 144
  utility functions 144
database access, TPFCS 165
database administrator (DBA) 150
database archives 166
database capacity 102
database file
  comparison with stream file 150
database identification (DBI) 136
database integrity 121
database ordinal number (DBON) 113, 114
  relationship to physical address 115
database reorganization 145
  pseudo module 145
database support 9
database utilities
  database reorganization 144, 145
  directory generation 148
  directory maintenance 148
  file capture and restore 144, 145
  file copy 144, 147
  file recoup 144, 147
  operation of 144

database utilities *(continued)*
pool directory generation 144
pool directory maintenance 144
deadlock 22, 23
deadlock detection 24
deferred list 61
define global fields macro (GLOBZ) 140
delay file 104, 139
demand paging 27
destination of message 171
device drivers 157
device type, logical
module file status table (MFST) 110
record duplication 121
directory
current directory 152
what is it? 151
working directory 152
directory reordering 129
dirty-reader protection, TPFCS 165
dispatch control list (DCL) 76
add an item to list 76
delete an item from list 76
dispatcher 61
dispatching 8
dispatching work 58, 76
dispensing pool record 123
Distributed Data Architecture (DDA) 150
duplication 102, 121
dynamic address translation (DAT) 26

E

ECB 69
ECB private area (EPA) 75
ECB register 70
ECB virtual memory
layout of 73
ECB virtual memory (EVM) 26, 72
ECB-controlled program 83
ECB-type program 83
element
access by keys 162
equality 163
ordering of collection 162
uniqueness of entries 163
emulator program (EP) 169
end user 3
ENTDC macro 80
enter a program and drop previous programs macro (ENTDC) 79
relationship with program nesting area 80
enter a program with no return expected macro (ENTNC) 79
enter program with expected return macro (ENTRC) 79
enter, definition of 80
enter/back processing
BACKC macro 80
enter a program and drop previous programs macro (ENTDC) 79
enter/back processing *(continued)*
enter a program with expected return macro (ENTRC) 79
enter a program with no return expected macro (ENTNC) 79
Enterprise Systems Connection (ESCON) 1
Entry 6, 39, 56
description of 80
reentrancy 41
relative to a process 39
termination of 92
entry control block (ECB) 6, 57, 69, 74
access 70
core block reference word (CBRW) 89
creation 70
database identification (DBI) 136
description of 67
ECB register 70
file address extension word (FAXW) 89
file address reference word (FARW) 89
format 70
program base identification (PBI) 136
relationship to Entry 56
size 70
environment
multiprocessing environment 42
uniprocessor environment 42
equipment, unit record 91
error handling, TPFCS 167
error recovery 92
ESA architecture 24
ESA configuration 1, 24
and multiprocessing 21
ESCON 1
example
path names 153
using symbolic link 155
exception recording 104
exception tape 145
executable script
starting a TPF application with 194
exit processing 92
expansion capability for database 129
extended globals 140
Extended Operations Console Facility/2 (EOCF/2) 92
external device support 166
external lock facility (XLF) 21, 32, 49, 109
description 144
lock maintenance 104
external symbol 79

F

fallback for pool 129
fast write 143
FIFO special file 158
file (write) a record 89
file accessibility 150
file address 105
file address compute (FACE) table (FCTB) 105, 130
database reorganization 145

file address compute (FACE) table (FCTB) *(continued)*
file layout 148
file address compute macro (FAC8C) 105
file address compute macro (FACZC) 105
file address compute program (FACE) 105
file address compute program (FACS) 105
file address extension word (FAXW) 89
file address reference format (FARF) 106
control bits 106
database reorganization 145
file address reference format 3 (FARF3) 106
file address reference format 4 (FARF4) 106
file address reference format 5 (FARF5) 106
file address reference format 6 (FARF6) 106
file address reference word (FARW) 89
file and unhold macro (FILUC)
relationship with record hold table 109
file capture and restore 104, 145
exception tape 145
full restore 145
load balancing 145
magnetic tape 145
partial restore 145
file copy database utility 147
file data chain transfer macro (FDCTC) 143
file ownership 150
file recoup database utility 147
file system
access permissions 150
file accessibility 150
file ownership 150
File Transfer Protocol (FTP) server
description of 189
socket application 189
FILNC macro, TPFCS 165
find (read) a record 89
find a file record macro (FINDC)
service routine 87
find and hold macro (FINHC)
processing in a loosely coupled environment 144
relationship with record hold table 109
fixed file 9
fixed file record 97
fixed record 99
initialization of a fixed record 148
initializing record ID 104
longevity 103
ordinal number 99
record ID 104
record type 99
reference to 106
use of 100
fixed record reference 106
fixed record type 99
fixed record type base address 115
fixed record type name 106
fixed storage 73, 74
fixed-file record 89
formatting a disk module 149
full restore 145
function management message router (FMMR) 182, 187
routing control parameter list (RCPL) 182

G

general data set (GDS) 9, 143
creation 143
file data chain transfer macro (FDCTC) 143
general file 9
creation 143
general tape 91
get file storage 128
get file storage macro 105
get file storage services 90
global area 42, 132
description 138
multiple database function (MDBF) 132
global directory 140
global field 140
global record 132
multiple database function (MDBF) 132
on a module 138
global synchronization 140
globals
data integrity 140
description 139
extended globals 140
global directory 140
global field 140
MDBF considerations 141
multiprocessing considerations 142
synchronize globals macro (SYNCC) 140
uses 139
group ID 150

H

hard link
comparison with symbolic link 156
what is it? 154
hardware storage arbitration 29
heap storage 75
MALOC 75
stack 75
High Performance Option (HPO) feature 98
home virtual address space 26
special use of 26
horizontal record allocation 9, 97
access by multiple Entries 97
Hypertext Transfer Protocol (HTTP) server
description of 189
socket application 188, 189

I

I-stream engine 21, 24
application I-stream engine 48, 49
categories of 48
CPU affinity 47
main I-stream engine 48

I-stream engine *(continued)*
moving work 84
multiprogramming 43
relation to SVM 26
shared record 132
switch I/O processing between I-stream engines 87
unique record 132, 133
I/O block (IOB) 74
use by FINDC macro service routine 88
I/O processing
switch I/O processing between I-stream engines 87
immediate file 104, 139
index, database organization 97
Information Management System (IMS) 169
initialization of a disk module 149
initializing a database 148
initializing TPFCS 167
input list 61
create new ECB and transfer control macro (CXFRC) 83
input message 3, 6, 11
destination 171
handled by non-SNA communications control 169
handled by SNA communications control 169
origin 169, 171
processing of 57
relationship to MIPS 16
instruction
compare and swap (CS) 30
compare double and swap (CDS) 30
contrast with command 32
interlock 31
load PSW (LPSW) 36
set system mask (SSM) 36
start subchannel (SSCH) 32
supervisor call (SVC) 36
test and set (TS) 30
integrity of database 121
integrity of message 182
Internet daemon
description of 188
socket application 188
interprocessor communications facility (IPC) 51, 182
interrupt 33
interrupt processing 33
concurrent interrupts 33, 34
hardware interrupt 36
relation to PSW 33
software interrupt 36
introduction to TPFCS 159
invoking a program 78
IPL virtual memory (IVM) 26, 72

K

key bag collection 163
key path support, TPFCS 162
key set collection 163
key sorted bag collection 163
key sorted set collection 163
keyed log collection 163
keypointing 8
keys, access 162
keys, element equality 163

L

limited lock facility (LLF) 32
line number, interchange address, terminal address (LNIATA) 171
link
comparison 156
hard 154
symbolic 155
what is it? 154
why use 154
linkage editor
LEDT 79
Nova LEDT 79
linkage editor (LEDT) 79
load balancing
communications control 180
file capture 145
load PSW instruction (LPSW) 36
loader general file 9, 79
loading data to the TPF system 149
local queue manager 184
lock 23
application lock 47
processor lock 44
spin lock 46
lock identity 32
lock indicator 29
lock, on a TPFCS collection 165
locking cursors 165
log collection 163
log processor 172, 178
logging 104, 189
logical device type 102
DEVA 102
DEVB 102
DEVC 102
DEVD 102
logical end-point identifier (LEID) 172
new application 172
old application 172
logical record cache and CF cache support 50
logical record cache support 50, 51
logical separation of data 98, 129
relation to subsystem user (SSU) 98
subsystem user (SSU) 129
logical storage block 74
logical unit (LU) 172
types of 186
long message transmission program (LMT) 136
long-term pool
return address 128
long-term pool record 103
loosely coupled
CF considerations 139
DASD record cache 143
database considerations 144

loosely coupled *(continued)*
external lock facility (XLF) 32, 49, 109
find and hold macro (FINHC) 144
global synchronization 140
in complex 181
interprocessor communication 38
interprocessor communications facility (IPC) 182
out of complex 181
processor unique record 134
record holding 109
shared module 32
system interprocessor global table (SIGT) 141
VFA considerations 139
loosely coupled complex 21
loosely coupled multiprocessing 21, 49
low entry network (LEN) interface (T2.1) 186

M

macro authorization 29
macro decoder 56
magnetic tape
file capture/restore 145
pilot tape 148
main I-stream engine 48
main storage 1
fixed storage 73
management of 72
maximum size 26
protection 28
retaining module records 137
working storage 73
main supervisor 8
management
main storage 72
mapping 118
message
flow through system 63
processing of 57
summary of flow through system 69
message destination
external 181
in complex 181
local 181
out of complex 181
output message 181
message origin 171
message routing, overview 170
MIPS 15
relationship to message 16
module device file copy 147
module file status table (MFST) 102, 131
purpose in system 110
symbolic device address (SDA) 110
symbolic module number 110
module record cache 142
move an Entry to another I-stream engine 86
moving work between I-stream engines 84
multi-path lock facility (MPLF) 32
Multi-Processor Interconnect Facility (MPIF) 38
multiple database function (MDBF) 98, 129
multiple database function (MDBF) *(continued)*
basic subsystem (BSS) 130
considerations for globals 141
global area 132
global record 132
module file status table (MFST) 131
reason to switch among subsystems and subsystem users 135
record ID attribute table (RIAT) 131
routing control application table (RCAT) 131
switching among subsystems and subsystem users 135
multiple I-stream DASD I/O 87
multiprocessing 14, 21
considerations for globals 142
deadlock 31
disadvantage 49
interprocessor communication 38
loosely coupled multiprocessing 21, 49
multiprogramming 43
reentrancy 42
summary 51
system interprocessor global table (SIGT) 141
tightly coupled multiprocessing 21, 44
multiprocessor environment
system evolution 42
multiprogramming 14, 21, 43
delay 43
in support of multiprocessing 43
reentrancy 42
summary 51
mutual exclusion 22
MVS
Customer Information Control System (CICS) 6
general data set (GDS) 143
general file 143
support environment for TPF 5

N

named pipe 158
network control program (NCP) 169, 175
network protocols
3270 local 186
airline lines control (ALC) 186
binary synchronous communication (BSC) 186
channel to channel (CTC) 186
synchronous data link control (SDLC) 186
synchronous link control (SLC) 186
token ring 186
X.25 186
network, simplistic 171
new application
logical end-point identifier (LEID) 172
resource identifier (RID) 172
node control block (NCB) 180
nonlocking cursor 165

O

obtaining a pool file address 128

old application
logical end-point identifier (LEID) 172
one application, simplistic network 171
online data loader 148
online loader (ACPL) 79
online system 5
operation of system 91
basic subsystem (BSS) 135
cycle down 139
optimistic concurrency 165
OPZERO 62, 67, 176
creating an ECB 58
functions 176
SNA 176
ordering of collection elements 162
ordinal number 9, 90, 99, 106
fixed record 106
usage 9
origin of message 171
output message transmission 182
overview of TPFCS 159

P

pacing 182
page 26
page 0 27
and the control program 80
uniqueness 27
page 0 reference 28
page table 26
paging 27
parallel processing 21, 22, 39
partial restore 145
path length 15, 17
path name
absolute path name 153
relative path name 153
what is it? 153
performance 17
data organization 95
data organization factors 95
design objective 8, 13
design of communications control 169
disadvantage 49
influenced by design of record allocation 102
performance considerations 49
record duplication 121
uniprocessor performance 49
permanently logged 177
pessimistic concurrency 165
physical separation of data 98, 129
relation to subsystem (SS) 98
subsystem (SS) 129
physical storage block 74
pilot tape 148
pipe 158
pool directory 99, 122
description 126
pseudo module 129
reordering 129
pool fallback 129
pool file storage 9, 99
database reorganization 145
directory 122, 126
directory generation 148
directory maintenance 148
directory reordering 129
dispensing pool record 123
expansion 145
fallback 129
file recoup 147
management of 124
ordinal number 123
pool directory generation 129
pseudo module 129
ratio dispensing 128
ratio factor 128
record type 123
recovering long-term pool records 147
release address 128
return address 128
section 124
segment 125
pool record 97, 99
initializing record ID 104
long-term 103
longevity 103
record ID 104, 105
record type 99
reference to 106
short-term 103
use of 100
pool record reference 106
pool record type 99
4K long-term (4LTx) 103
4K long-term duplicated (4DPx) 103
4K short-term (4STx) 103
large long-term (LLTx) 103
large long-term duplicated (LDPx) 103
large short-term (LSTx) 103
small long-term (SLTx) 103
small long-term duplicated (SDPx) 103
small short-term (SSTx) 103
pool record types 103
pool section
description 124
fallback 129
short-term pool recycling 129
pool segment
description 125
pools 90
post interrupt (PI) vector
add work to specified I-stream macro ($ADPC) 87
dispatch control list 77
prefix register 27
prefixing 27
primary record 102, 121
primary virtual address space 26
process
abstraction of an Entry 39, 108
concurrent processes 22, 54

process *(continued)*
contrast with program 39
sequential process 22
simultaneous processes 22, 54
process models
AOR 189
DAEMON 188
NOLISTEN 189
NOWAIT 188
RPC 189
WAIT 188
processing an input message 57
processing center 2
processing of an entry is complete macro (EXITC) 69, 92
processor
shared record 132
unique record 132, 134
processor lock 44
program 39
application program 39, 83
contrast with process 39
critical region 42
delay 43
E-type program 83
ECB-controlled program 83
program segment 40
reentrant program 39, 42
serially reusable program 42
system ECB-controlled program 83
system program 39
transaction program 183
program allocation table 79
program animation 108
program base identification (PBI) 136
program classification 80
control program 80
ECB-controlled program 83
program fetch 58
program linkage 78
program nesting 80
program nesting area 80
program status word (PSW) 33
relation to interrupt processing 33
property service functions 164

Q

queueing time 112

R

ratio dispensing 128
ratio factor 128
ready list 61
create new ECB and transfer control macro (CXFRC) 83
real address 26
real-time tape 90
reconstruction support 166
record accessing 106
record address conversion services 105
record allocation
database ordinal number (DBON) 113
example of 114
horizontal allocation 9, 97
vertical allocation 9, 97
record duplication 102, 121
record hold table 44
lock maintenance 104
relation to XLF lock table 144
use of 108
record holding 108, 144
loosely-coupled complex, in a 109
record ID 104, 106
parameter in file address reference word (FARW) 104
record ID attribute table (RIAT) 104
record ID attribute table (RIAT) 104, 131
DASD record caching candidacy characteristics 143
exception recording characteristics 104
lock maintenance characteristics 104
logging characteristics 104
module record caching candidacy characteristics 104
user exit characteristics 104
VFA candidacy characteristics 104, 138
record longevity 103
record reference 105
fixed record 106
pool record 106, 123
record sharing table (RST) 139
record size 102
record type 9, 89
fixed record 99, 106
pool record 99
usage 9
recovering long-term pool records 147
recovery of message 182
recycling short-term pool 128, 129
reduction, data 59
reentrancy 41
Entry 41
multiprocessing 42
multiprogramming 42
use of a stack 81
reentrant program 39, 42
contrast with serially reusable program 39
RENT compile-time option 39
writable static 39
relative path name 153
request system services 58
resource identifier (RID)
new application 172
resource vector table (RVT) 178
response time 1, 6, 14
restore 104
retentive write 143
retrieving Web pages 193
return file pool address 128
return file pool address macro (RELFC) 123
return to previous program record macro (BACKC) 80

roll call 110
rollback transaction 12
route a message macro (ROUTC) 182
  presentation services 183
routing control application table (RCAT) 131, 178
  use by switch entry to another I-stream engine macro (SWISC) 86
routing control block (RCB) 180
routing control parameter list (RCPL) 68, 171, 177, 178
  construction of 178
  function management message router (FMMR) 182
  terminal control block 181

S

scratchpad area (SPA) 180
seek time 112
segment table 26
sending output message 181
sequence collection 163
sequential processing 55
serialization 30
serially reusable 84
serially reusable program 42
  contrast with reentrant program 42
set collection 163
set system mask instruction (SSM) 36, 55
shadowing support 166
shared DASD
  CF record lock support 32
  locking 32
shared data
  CPU affinity 47
shared record 132
  example of subsystem user (SSU) 132
  I-stream engine 132
  processor 132
  subsystem user (SSU) 132
short-term pool
  recycling 128, 129
  return address 128
short-term pool record 103
short-term pool recycling 128, 129
simultaneous processes 22, 54
SNA network
  data host 186
  interface 186
  low entry network (LEN) interface (T2.1) 186
  subarea interface (T5) 186
SNA networks 172
socket application
  File Transfer Protocol (FTP) server 189
  HTTP server 189
  Internet daemon 188
  RPC server 190
  syslog daemon 189
  TFTP server 189
sorted bag collection 164
sorted set collection 164
special files 157
spin lock 31, 46
stack 81
stack storage 75
start subchannel instruction (SSCH) 32
starting a TPF application
  from the Internet 194
storage protection 28
storing Web page contents 192
storing Web pages 192
stream file
  comparison with database file 150
  using in programs 151
  what is it? 150
  why use 150
subarea interface (T5) 186
subsystem (SS) 98
  basic subsystem (BSS) 130
  physical separation of data 129
  relation to subsystem users (SSU) 130
  switching among subsystems 135
subsystem user (SSU) 98
  logical separation of data 129
  relation to subsystem (SS) 130
  shared record 132
  switching among subsystem users 135
  unique record 132, 133
subsystem user ID (SSU ID) 136
  subsystem user ID (SSU ID) 136
subsystem user shared record 132
supervisor call instruction (SVC) 36
suspend list 61
suspend processing for ECB I/O completion macro (WAITC) 89
switch entry to another I-stream macro (SWISC)
  create option 88
  dispatching from ready list 86
  enter option 79
  input parameters 86
  processing description 86
  service routine 86
switch I-stream engine for application processing 179
switch I/O processing between I-stream engines 87
symbolic address 9
symbolic device address (SDA) 110
symbolic link
  comparison with hard link 156
  example of using 155
  what is it? 155
symbolic module number 110
synchronization of globals 140
synchronization, parallel processing 22
synchronize globals macro (SYNCC) 140
synchronous data link control (SDLC) 186
synchronous link control (SLC) 176, 186
syslog daemon
  description of 189
system allocator (SAL) 79
system allocator table 79
system heap 75, 76
system initialization 60
system interprocessor global table (SIGT) 141

system loader 79
system services
  record address conversion 105
system services request 58
system state 35
system test compiler program (STC) 148
system virtual memory
  layout of 73
system virtual memory (SVM) 26, 72
  relation to I-stream engine 26
system work block (SWB) 74

T

tape status table (TSTB) 91
tape, magnetic 90
  general tape 91
  real-time tape 90
  symbolic addressing 91
TCP/IP
  See Transmission Control Protocol/Internet Protocol (TCP/IP)
terminal address table (WGTA) 178
terminal concentrator 3
terminal control block 180
  agent assembly area (AAA) 180
  node control block (NCB) 180
  routing control block (RCB) 180
  routing control parameter list (RCPL) 181
  scratch pad area (SPA) 180
terminal identification table 178
  resource vector table (RVT) 178
  terminal address table (WGTA) 178
termination of an Entry 92
test and set instruction (TS) 29, 30, 46
throughput 15
tightly coupled
  database considerations 144
  interprocessor communication 38
  multiprocessor 21
  uniprocessor 21
tightly coupled multiprocessing 21, 44
token ring 186
TPF Advanced Program-to-Program Communications (TPF/APPC) 183
  presentation services 183
TPF Application Requester (TPFAR) 38, 150
TPF collection support
  APIs 167
  archiving support 166
  benefits 159
  browse support 168
  capture and restore support 166
  concurrency controls 165
  database access 165
  database archives 166
  error handling 167
  key path support 162
  overview 159
  properties, collection 164
  reconstruction support 166
TPF collection support *(continued)*
  shadowing support 166
  supported collections 163
  TPF transaction services 166
  validation support 166
  ZBROW commands 168
  ZOODB commands 167
TPF Database Facility (TPFDF)
  database administrator (DBA) 150
  Distributed Data Architecture (DDA) 150
TPF Internet mail server support
  description of 190
  IMAP server 190
  POP server 190
  SMTP server 190
TPF Internet server support
  description of 191
TPF MQSeries support
  local queue manager 184
TPF transaction services
  commit scope 12
    begin 12
    commit 12
    resume 12
    rollback 12
    suspend 12
  log manager 13
  recovery log 13
  resource manager 12
  transaction manager 12
transaction 6, 11
  definition 11
  editor 180
  manager 12
Transaction Processing Facility (TPF)
  applicability 5
  backup 7
  characteristics 6
  control program 56
  control structure 56
  data host 186
  design 59
  history 3
  interrupt handling 35
  performance 17
  problem of system evolution 42
  recovery 7
  supporting environment 7
transaction program 183
transaction program name table (TPNT) 183
transfer of control 83
transfer time 112
Transmission Control Protocol/Internet Protocol 169
Transmission Control Protocol/Internet Protocol (TCP/IP) 187
Trivial File Transfer Protocol (TFTP) server
  description of 189, 190
  socket application 188, 189, 190

U

unhold file record macro (UNFRC)
  relationship with record hold table 109
uniprocessor 21
uniprocessor environment
  system evolution 42
  uniprocessor performance 49
unique record 132
  I-stream engine 133
  processor 134
  subsystem user (SSU) 133
uniqueness of entries in collections 163
unit record equipment 91
update sequence counters 165
user ID 150

V

V-con 79
validation support 166
VCT (Virtual file access count) list 61
vertical record allocation 97
VFA buffer 138
via a page 0 reference 28
virtual address 26
virtual address space 72
virtual addressing 25, 27
virtual file access (VFA) 138
  aging out process 139
  buffer 138
  candidacy characteristics 104
  delay file 104, 139
  description 138
  immediate file 104, 139
  loosely coupled considerations 139
  record ID attribute table (RIAT) 104
  user exit 139
virtual file access count (VCT) list 61
virtual memory
  ECB virtual memory (EVM) 26, 72
  IPL virtual memory (IVM) 26, 72
  system virtual memory (SVM) 26, 72
virtual storage
  layout of 73
Virtual Telecommunications Access Method (VTAM)
  communications management configuration (CMC) 186

W

Web pages
  retrieving 193
  storing 192
work scheduling 8
working storage 73, 74
  4K common frame 74
  4K frame 74
  entry control block (ECB) 74
  heap storage 75
  I/O block (IOB) 74
working storage *(continued)*
  logical storage block 74
  physical storage block 74
  relationship of common frame to common block 75
  relationship of frame to logical block 74
  system work block (SWB) 74
write PIU to NCP 37x5 macro (SOUTC) 182

X

X.25 186
XLF lock table 144
  relation to record hold table 144

Z

ZBROW commands 168
ZOODB commands 167

File Number: S370/30XX-20
Program Number: 5748-T14

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GH31-0139-12