Redbooks
ibm.com/redbooks

ABCs of z/OS System Programming Volume 1

Lydia Parziale

Luiz Fadel

Stanley Jon

IBM Z

IBM

Redbooks

International Technical Support Organization

ABCs of z/OS System Programming Volume 1

November 2017

SG24-6981-04

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

Fifth Edition (November 2017)

This edition applies to version 2 release 3 of IBM z/OS (product number 5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

Preface

The ABCs of IBM® z/OS® System Programming is a 13-volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects.

Whether you want to become more familiar with z/OS in your current environment, or you are evaluating platforms to consolidate your online business applications, the ABCs collection serves as a powerful technical tool.

Volume 1 provides an updated understanding of the software and IBM Z architecture, and explains how it is used together with the z/OS operating system. This includes the main components of z/OS needed to customize and install the z/OS operating system. This edition has been significantly updated and revised.

The other volumes contain the following content:

- Volume 2: z/OS implementation and daily maintenance, defining subsystems, IBM Job Entry Subsystem 2 (JES2) and JES3, link pack area (LPA), LNKLST, authorized libraries, System Modification Program/Extended (SMP/E), and IBM Language Environment®
- Volume 3: Introduction to Data Facility Storage Management Subsystem (DFSMS), data set basics, storage management hardware and software, catalogs, and DFSMS Transactional Virtual Storage Access Method (VSAM), or DFSMStvs
- Volume 4: z/OS Communications Server, Transmission Control Protocol/Internet Protocol (TCP/IP), and IBM Virtual Telecommunications Access Method (IBM VTAM®)
- Volume 5: Base and IBM Parallel Sysplex®, z/OS System Logger, Resource Recovery Services (RRS), Global Resource Serialization (GRS), z/OS system operations, z/OS Automatic Restart Manager (ARM), IBM Geographically Dispersed Parallel Sysplex™ (IBM GDPS®)
- Volume 6: Introduction to security, IBM Resource Access Control Facility (IBM RACF®), Digital certificates and public key infrastructure (PKI), Kerberos, cryptography and IBM eServer™ z990 integrated cryptography, zSeries firewall technologies, Lightweight Directory Access Protocol (LDAP), and Enterprise Identity Mapping (EIM)
- Volume 7: Printing in a z/OS environment, Infoprint Server, and Infoprint Central
- Volume 8: An introduction to z/OS problem diagnosis
- Volume 9: z/OS UNIX System Services
- Volume 10: Introduction to IBM z/Architecture®, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and IBM DS8000®
- Volume 11: Capacity planning, IBM Performance Management, z/OS Workload Manager (WLM), IBM Resource Management Facility (IBM RMF™), and IBM System Management Facility (SMF)
- Volume 12: WLM
- Volume 13: JES2 and JES3 System Display and Search Facility (SDSF)

Authors

This book was produced by a team of specialists from around the world working at the IBM International Technical Support Organization (ITSO), Poughkeepsie Center.

Lydia Parziale is a Project Leader for the ITSO team in Poughkeepsie, New York, with international experience in technology management including software development, project leadership, and strategic planning. Her areas of expertise include business development and database management technologies. Lydia is a certified PMP and an IBM Certified IT Specialist with an MBA in Technology Management. She has been employed by IBM for over 25 years in various technology areas.

Luiz Fadel is a consultant for Maffei Consultoria em Informatica Ltda. He retired from IBM in 2013 as an IBM Distinguished Engineer responsible for supporting IBM System z® for the Latin America region, part of the Growth Markets Unit. He joined IBM in 1969 and has supported large systems ever since, including working on two assignments with the International Technical Support Organization (ITSO). Luiz is a member of the Latin America Advanced Technical Support team, which is responsible for handling Client Critical Situations and client claims within IBM Z, Early Support Programs, new product installations, internal product announcements, and second-level client support, as well as managing complex proofs of concepts (POCs). He is a member of the zChampions team and the co-author of several IBM Redbooks® publications.

Stanley Jon is a z/OS IT Specialist in Canada. He has been with IBM Canada and working on the IBM Z platform for 20 years. He started in the IBM Support Centre supporting z/OS and ICSF and is now working in the field assigned to a Canadian financial institution. His areas of expertise include z/OS and cryptography on the mainframe.

Thanks also to the following contributors to this edition:

Robert Haimowitz, William G. White
International Technical Support Organization, Poughkeepsie Center

Rafael Lima, AMS Mainframe IBM MVS™ Software
DXC Technology

Albert P. Franco II, Enterprise IT Services
State of Nevada, Department of Administration

Thanks to the authors of the fourth edition of this book:

Karan Singh, Paul Rogers

Now you can become a published author, too

Here’s an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time. Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run 2 - 6 weeks in length, and you can participate either in person or as a remote resident working from your home base.

Obtain more information about the residency program, browse the residency index, and apply online at the following website:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us.

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form:

  ibm.com/redbooks

- Send your comments in an email:

  redbooks@us.ibm.com

- Mail your comments:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

The World of IBM Z

IBM Z is a highly secure, connected, and cognitive platform. It is also known for its reliability, scalability, and availability, so the Z mainframe manages a large percentage of the world's mission critical business data and transactions.

Z is a mature environment, and its wide breadth of features and functions can pose a daunting challenge for new users and system programmers to learn and be productive in their roles. The ABCs of IBM z/OS System Programming series is intended to educate and act as a reference to newcomers to the mainframe world. This chapter provides a quick overview of the core hardware and software components of the Z environment.

This chapter covers the following topics:

- IBM Z hardware configuration
- Parallel Sysplex
- Coupling facility
- Cryptographic hardware
- z/OS services
- ICSF
- z/OS security
- Data Facility Storage Management Subsystem
- IBM Health Checker for z/OS
- IBM z/OS Management Facility

1.1 IBM Z hardware configuration

A typical Z hardware configuration consists of the mainframe processor, system consoles, storage devices (DASD and tape), and network connectivity. Each mainframe processor must be logically partitioned into smaller systems, called *logical partitions* (LPARs). An LPAR runs an operating system, such as z/OS, IBM z/VM®, Linux for System z, or IBM z/VSE®, and has access to I/O devices. See Figure 1-1. For more information about Z hardware, see *ABCs of z/OS System Programming Volume 10*, SG24-6990.

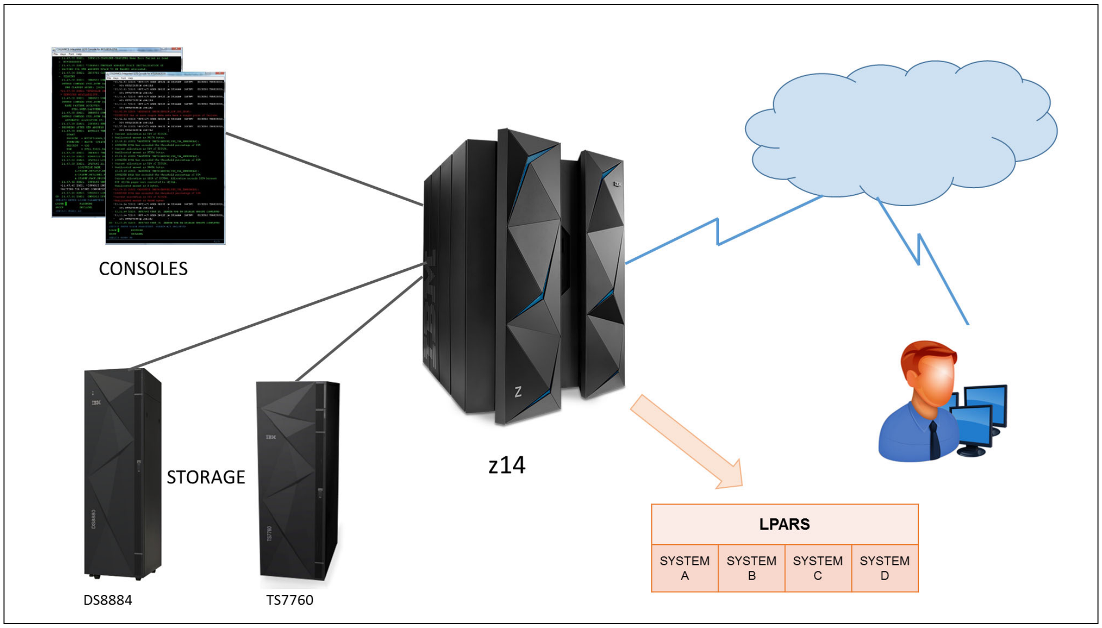

Figure 1-1 Typical IBM Z environment

1.2 Parallel Sysplex

The configuration, known as a *system complex* or SYSPLEX, is composed of several mainframe systems clustered together and connected by channel-to-channel (CTC) links for peer-to-peer communications. When a coupling facility (CF) is part of this configuration, it is known as a *Parallel Sysplex*.

As shown in Figure 1-2, a Parallel Sysplex provides higher availability and performance because the workload can be shared among the systems.

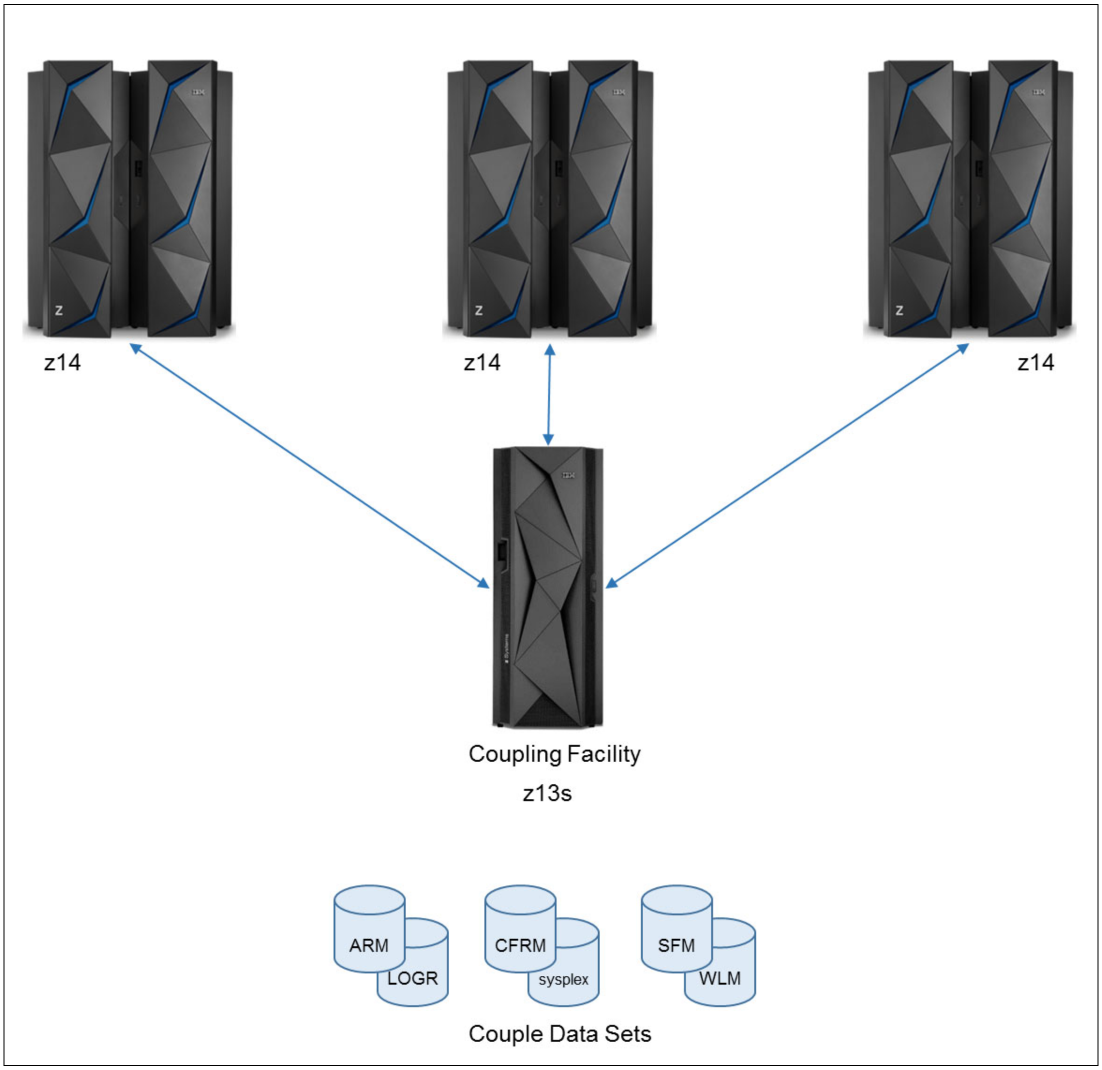

Figure 1-2 Parallel Sysplex

1.3 Coupling facility

A coupling facility (CF) is a processor that runs a specialized operating system known as Coupling Facility Control Code (CFCC). The CF only contains control blocks known as structures which are used by mainframe subsystems and applications.

1.3.1 CF structure encryption

To improve protection against potential security breaches that could otherwise expose sensitive data, IBM z/OS V2.3 supports encryption on all data being transmitted between z/OS systems and the CFs and when the data is at rest in the CF. The cryptography is transparent to all CF exploiters, so no changes to subsystem environments or applications itself are necessary.

How it works

All of the cryptography is done by a z/OS service called *Cross-system Extended Services* (XES). XES uses the cryptographic hardware on the mainframe to encrypt and decrypt the data with Advanced Encryption Standard (AES) protected keys. The encryption of a CF structure is controlled by the Coupling Facility Resource Management (CFRM) policy with the new structure parameter **ENCRYPT**.

XES encrypts the data and sends it along the CF link to the CF (see Figure 1-3). When the application needs the data, the encrypted data is transferred from the CF to XES to be decrypted and then passed back to the application (see Figure 1-4). No encryption or decryption is done by the CF or by the CF links.

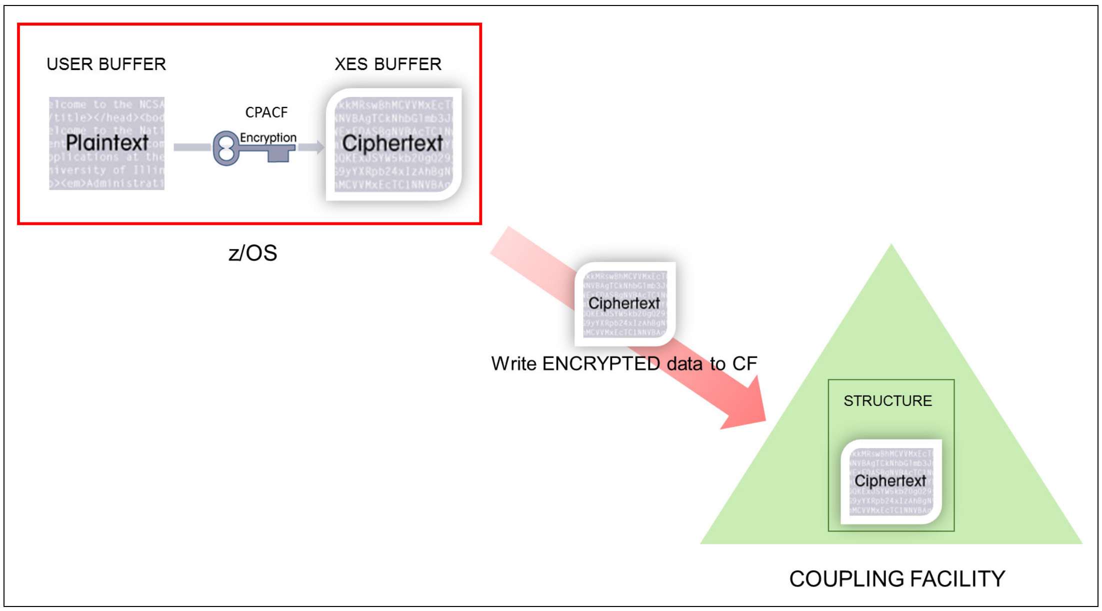

Figure 1-3 Write encrypted data to CF

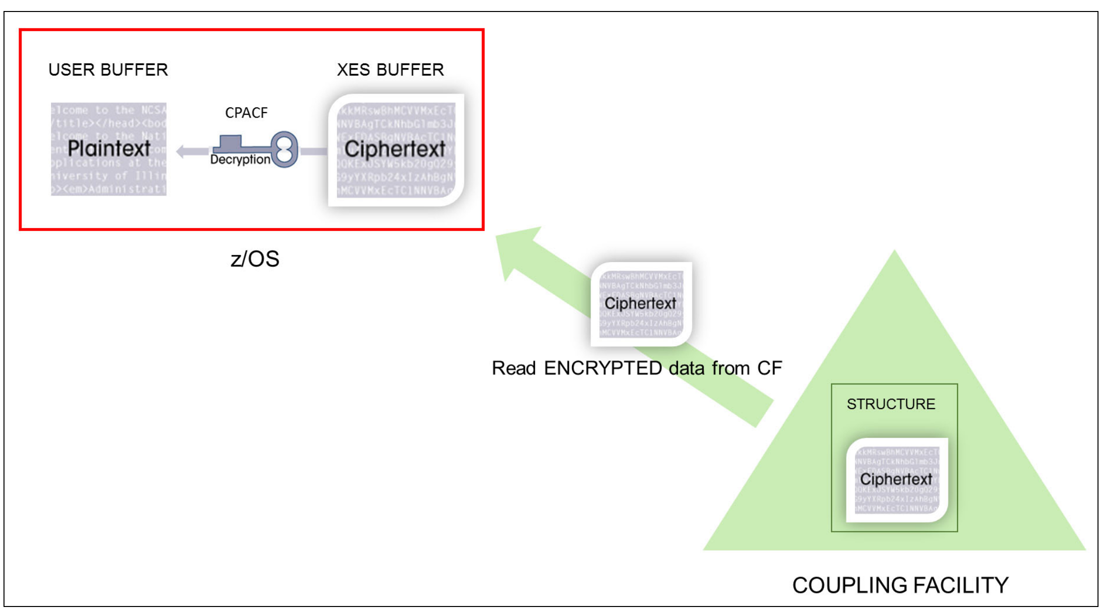

Figure 1-4 Read encrypted data from CF

Requirements

The minimum hardware requirement for this function is Crypto Express3 and CPACF. The minimum software requirements are z/OS V2.3 and ICSF HCR77C0. Also, the AES Master Key and the ICSF Cryptographic Key Data Set (CKDS) must be the same in the entire sysplex.

1.4 Cryptographic hardware

There are two cryptographic hardware features available on all Z mainframe processors. The two features are the Central Processor Assist for Cryptographic Function (CPACF) and IBM Crypto Express.

CPACF is a cryptographic feature that performs hashing, random number generation and symmetric cryptography with clear keys and protected keys. This feature is available on all general-purpose processors in the mainframe and has been improved on IBM z14 in order to better support the pervasive encryption objective. CPACF on z14 is six times faster than IBM z13® when encrypting the same size of data and using the same cryptographic mode of operation.

The Crypto Express feature performs secure key symmetric and asymmetric cryptography with Crypto Express6S available on z14 and Crypto Express5S available on z13. Both Crypto Express features can be configured as a CCA cryptographic coprocessor, as an accelerator, or as a PKCS #11 cryptographic coprocessor. On Crypto Express6S, the performance of Public Key Cryptography (asymmetric cryptography) and the secure module tamper detection has been enhanced.

Keep in mind that the cryptographic hardware available on z13 can be used to implement all the pervasive encryption initiatives available in z/OS V2.3 (or on z/OS V2.2 with the requisite APARs). However, z14 is the suggested hardware level, because it provides faster encryption of data in-flight and at-rest with enhanced on-chip cryptographic performance and a faster Crypto Express6S.

1.5 z/OS services

z/OS is a 64-bit operating system and provides program management services that enable you to create, load, modify, list, read, and copy executable programs.

IBM delivers a z/OS release every two years. See Table 1-1 for the general availability dates of three z/OS releases. New z/OS functions continue to be delivered between releases through the normal maintenance stream, or as web deliverables. In addition, significant new functions might be delivered between releases as features of the product.

Table 1-1 General availability dates for z/OS

Product	General Availability
z/OS V2R3	September 30, 2017
z/OS V2R2	September 30, 2015
z/OS V2R1	September 30, 2013

The z/OS system provides solutions for the following major areas, as illustrated in Figure 1-5:

- **Data management:** z/OS provides a set of functions to support the following tasks:
  - Manage storage resources on the system.
  - Support storage and retrieval of data on disk, optical, and tape devices.
  - Offer program management functions.
  - Supply device management functions to define and control the operation of input and output storage devices.

  Distributed FileManager (DFM) supports access to remote data and storage resources.
- **Softcopy publications services:** These services improve productivity in systems installation and management.
- **Security services:** Security and Cryptographic Services are a set of products and features used to control access to resources, and to audit and manage the accesses with appropriate centralized or decentralized control. These services form the basis for all security services for traditional applications, UNIX applications, and distributed systems.

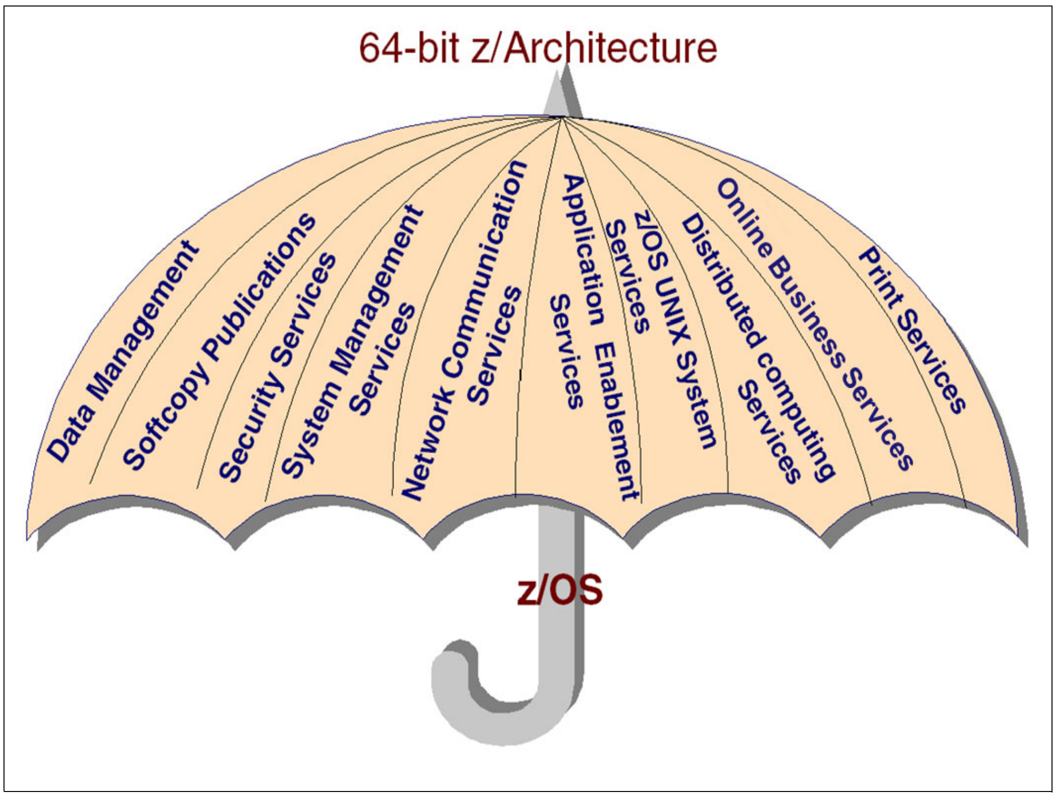

Figure 1-5 z/OS services

- **System management services:** The functions and features provided with z/OS support robust control and automation of the basic processes of z/OS. This functionality increases availability, improves productivity of system programmers, and provides a consistent approach for configuring z/OS components of products.

- **Network communication services:** z/OS enables world-class Transmission Control Protocol/Internet Protocol (TCP/IP) and Systems Network Architecture (SNA) networking support, multivendor and multiplatform connectivity, connectivity to a broad set of users, and support for multiple protocols.
- **Applications enablement services:** These services provide a solid infrastructure in which you can build new applications, extend existing applications, and run online transaction processing (OLTP) and batch processes.
- **z/OS UNIX system services:** z/OS contains the UNIX applications services (shell, utilities, and debugger) and the UNIX System Services (kernel and runtime environment). The shell and utilities provide the standard command interface familiar to interactive UNIX users. z/OS includes all the commands and utilities specified in the X/Open Portability Guide (XPG) 4.2.

  With IBM Language Environment, z/OS supports industry standards for C programming, shell and utilities, client/server applications, and the majority of the standards for thread management. This functionality enables transparent data exchange and easy portability of applications in an open environment.
- **Distributed computing services:** These services are achieved by a set of features and functions. z/OS Network File System (NFS) acts as a file server to workstations, personal computers, or other authorized systems in an Internet Protocol network. Remote files are mounted from the mainframe (z/OS) to appear as local directories and files on the client system. DCE enables Data Encryption Standard (DES) algorithms and the commercial data masking facility (CDMF).

  Distributed File Services (DFS) system-managed buffering (SMB) enables users to access data in a distributed environment across a wide range of IBM and non-IBM platforms. SMB can automatically handle the conversion between American Standard Code for Information Interchange (ASCII) and Extended Binary Coded Decimal Interchange Code (EBCDIC).
- **Online business services:** The IBM Hypertext Transfer Protocol (HTTP) Server provides for scalable, high-performance web serving for critical online business applications. It is exclusive to z/OS. Beginning in z/OS V2.2, this element is now IBM HTTP Server Powered by Apache.
- **Print services:** Application output can be electronically distributed and printed, or presented over the web.

1.6 ICSF

Integrated Cryptographic Service Facility (ICSF) is the z/OS cryptographic component that is the software interface to the Crypto Express hardware on the Z mainframes. You use ICSF to manage the cryptographic environment and hardware. Access to the cryptographic services available on the hardware is through a set of ICSF APIs. There are two types of cryptography, and ICSF supports both types through its APIs. The two types of cryptography are *symmetric* and *asymmetric* cryptography.

Symmetric cryptography is also known as “secret” key cryptography because you use the same key to encrypt and decrypt the data (see Figure 1-6 on page 8). For the data to be secure, you need to keep the key secure. Some well-known symmetric algorithms include Data Encryption Standard (DES) algorithm and Advanced Encryption Standard (AES).

Figure 1-6 shows an example of symmetric cryptography.

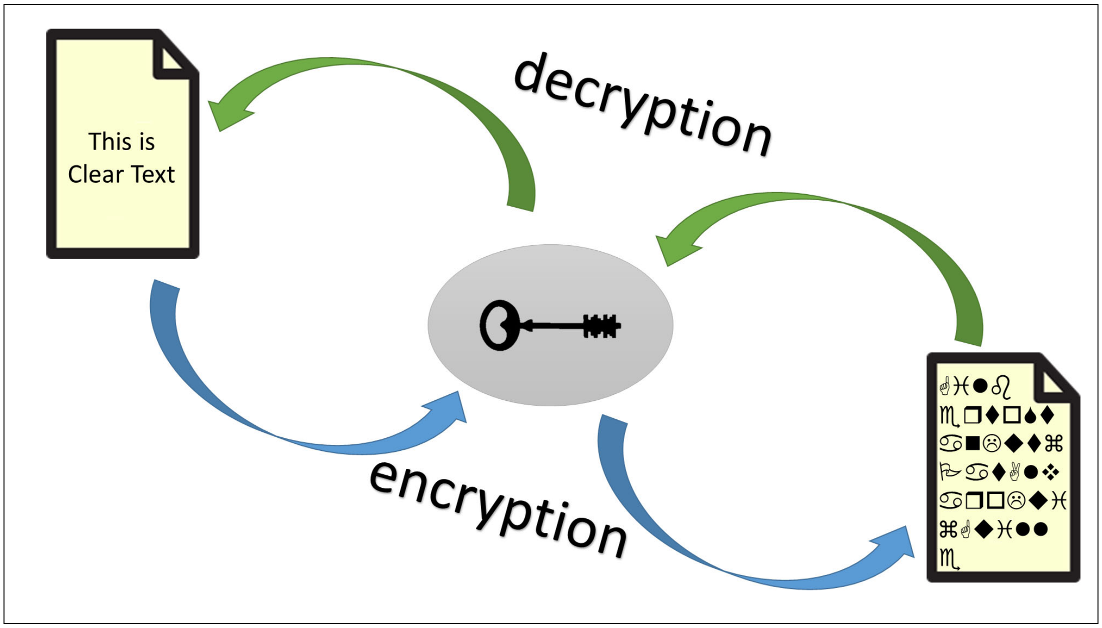

Figure 1-6 Symmetric cryptography

Asymmetric cryptography is also known as public key cryptography. In asymmetric cryptography, a public and private key pair is generated. The public key is used to encrypt data and the respective private key is used to decrypt the data (see Figure 1-7). The public key does not have to be kept secret because it cannot be used for decryption, whereas the private key does need to be protected because it is used to decrypt data. Popular asymmetric algorithms are the RSA algorithm and Elliptic Curve Cryptography (ECC).

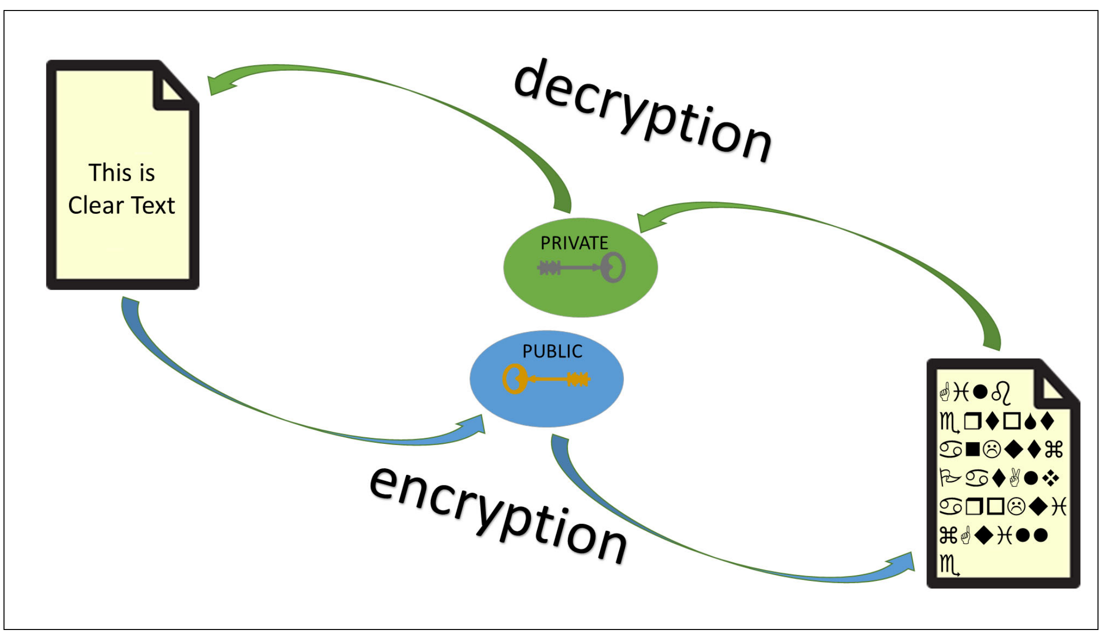

Figure 1-7 Asymmetric cryptography

Keys and tokens used by the ICSF cryptographic services are kept securely in three separate repositories known as key data sets. The three key data sets are:

Cryptographic Key Data Set (CKDS) Stores DES, AES, and HMAC keys.

PKA Cryptographic Key Data Set (PKDS) Stores RSA and ECC key pairs.

Token Data Set (TKDS) Stores PKCS#11 tokens and objects.

The CKDS and PKDS are both secure repositories because the keys stored in both data sets are encrypted by the ICSF master keys. There are four different master keys in ICSF and they are:

DES Master Key Encrypts the DES keys stored in the CKDS.

AES Master Key Encrypts the AES and HMAC keys stored in the CKDS.

RSA Master Key Encrypts the RSA private key stored in the PKDS.

ECC Master Key Encrypts the ECC private key stored in the PKDS.

All the Master Keys are stored in registers in the tamperproof Crypto Express hardware. The Master Keys are entered using a Trusted Key Entry (TKE) workstation or the ICSF dialogs.

The Z cryptographic environment is illustrated in Figure 1-8.

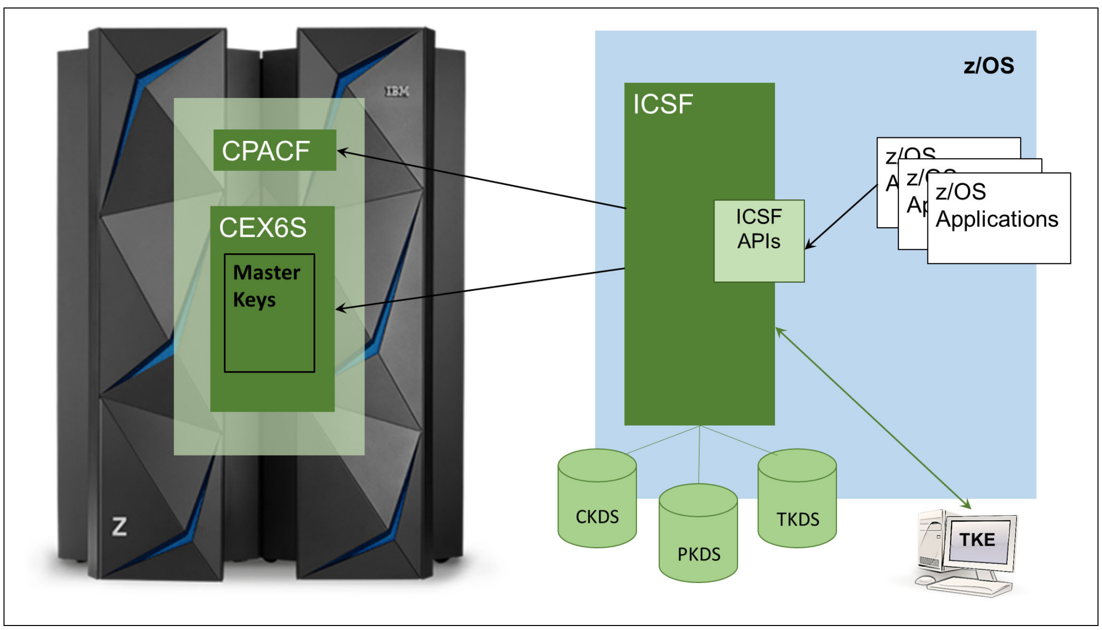

Figure 1-8 Z cryptographic environment

Types of keys used by ICSF

There are three key types that are used in the Z cryptographic environment.

Clear Key A cryptographic data key unprotected in the environment. For example, if you perform any type of dump of the data at the appropriate time, you can find the key in the dump. CPACF uses clear keys as input for its cryptography.

Secure Key A cryptographic data key that is protected in the environment by the master key. If you perform any type of dump of the data, you can only find the encrypted key in the dump. The Crypto Express card uses secure keys as input for its cryptography.

Protected Key A hybrid of a clear and a secure key. A protected key is a secure key decrypted within the secure Crypto Express hardware and re-encrypted or “wrapped” with the LPAR’s wrapping key. The protected key is passed back to ICSF, which is now eligible for CPACF operations. The advantage of using a protected key is that it leverages the speed of the CPACF feature with more security than a clear key.

1.7 z/OS security

As general computer literacy and the number of people using computers has increased, the need for data security has taken on a new level of importance. The installation can no longer depend on keeping data “secure” simply because no one knows how to access it. Further, making data secure does not mean only making confidential information inaccessible to those without permission to see it; it also means preventing the inadvertent destruction of files by people who might not even know that they are improperly manipulating data.

1.7.1 SAF

All z/OS components and subsystems use the System Authorization Facility (SAF) interface to validate access and authorization to resources. SAF is a component within z/OS, and does not require any other product as a prerequisite. However, it is extremely rare for any installation not to use an external security manager, because overall system security functions are greatly enhanced and complemented if it is used concurrently with RACF or another external security manager.

All of the SAF calls are processed by the z/OS router. To use the z/OS router, the z/OS component or subsystem issues the `RACROUTE` macro. When it is started, the z/OS router first calls an optional installation exit routine and then calls the external security manager (such as RACF), if one is active and installed on the system. See Figure 1-9.

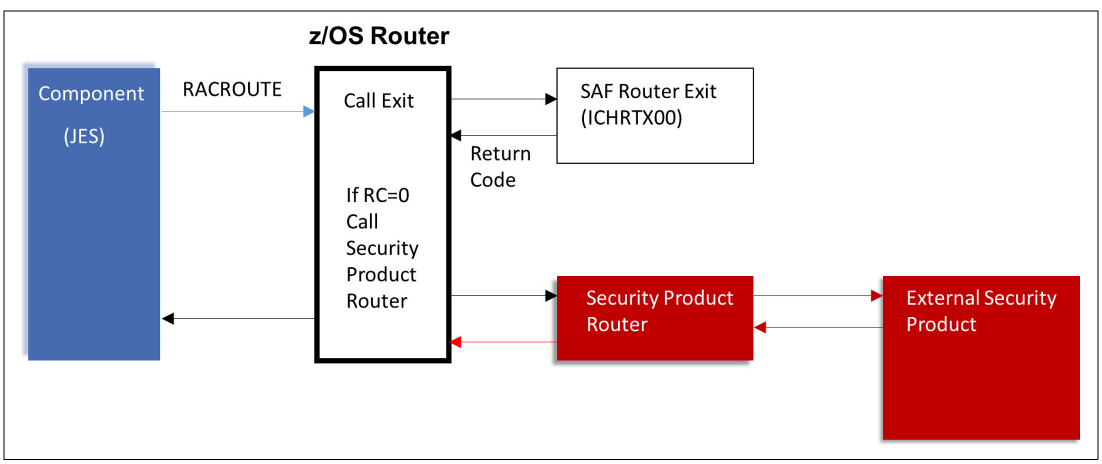

Figure 1-9 RACROUTE processing

1.7.2 RACF

The z/OS Security Server is the IBM *security product*. The RACF product is a *component* of the z/OS Security Server, and it works together with the existing system features of z/OS to provide improved data security for an installation. If this product is to be installed in your environment, then RACF customization must be done.

RACF helps meet the need for security by providing the following functionality:

- Flexible control of access to protected resources
- Protection of installation-defined resources
- Ability to store information for other products
- Choice of centralized or decentralized control of profiles
- An ISPF panel interface
- Transparency to users
- Exits for installation-written routines

RACF security protection

RACF controls access to and protects resources. For a software access control mechanism to work effectively, it must first identify the person who is trying to gain access to the system, and then verify that the user is really that person.

RACF uses a user ID in a user profile in the RACF database, and a system-encrypted password or pass phrase, to perform its user identification and verification. When you define a user to RACF, you assign a user ID and temporary password. The user ID identifies the person to the system as a RACF user. The password or pass phrase verifies the user's identity. The temporary password permits initial entry to the system, at which time the person is required to choose a new password.

RACF authorization checks

Having identified a valid user, the software access control mechanism must next control interaction between the user and the system resources. It must authorize what resources that user can access, but also in what way the user can access them, such as for reading only, or for updating and reading. This controlled interaction, or authorization checking, is done by RACF. However, before this activity can take place, someone with the appropriate authority at the installation must establish the constraints that govern those interactions.

With RACF, you are responsible for protecting the system resources (data sets, tape and DASD volumes, IBM IMS™ and IBM CICS® transactions, TSO logon information, and terminals) and for issuing the authorities by which those resources are made available to users. RACF records your assignments in profiles stored in the RACF database. RACF then refers to the information in the profiles to decide whether a user is to be permitted to access a system resource.

1.8 Data Facility Storage Management Subsystem

The Data Facility Storage Management (DFSMS) subsystem consists of a suite of related data and storage management products for the z/OS system. DFSMS is an operating environment that helps automate and centralize the management of storage based on the policies that your installation defines for availability, performance, space, and security.

DFSMS consists of one z/OS base element and four z/OS features:

- **DFSMSdfp** is *Data Facility Product* and is part of the z/OS base elements. Together with the Base Control Program (BCP), it forms the foundation of the z/OS operating system, performing the essential data, storage, and device management functions of the system.
- **DFSMSdss** is *Data Set Services* (dss) and is a DASD data and space management tool.
- **DFSMShsm** is *Hierachical Storage Manager* (hsm) and is a DASD storage management and productivity tool for managing low-activity and inactive data.

- **DFSMSrmm** is *Removable Media Manager* (rmm) and helps you manage your removable media as one enterprise-wide library across z/OS systems that can either share DASD or have TCP/IP connectivity.
- **DFSMStvs** is *Transactional VSAM Services* (tvs) and enables batch jobs and CICS online transactions to update shared VSAM data sets concurrently.

Except for DFSMSdfp, which is a base element, all others features are exclusive and optional. For more information about DFSMS, see *ABCs of z/OS System Programming Volume 3*, SG24-6983.

1.8.1 z/OS Data Set Encryption

As part of the pervasive encryption initiative, DFSMS has been enhanced to provide data set encryption in z/OS V2.3 and z/OS V2.2 with the required APARs. The data set encryption feature is a simple and transparent approach to enable extensive encryption of data at rest using DFSMS access methods. As with CF structure encryption, DFSMS uses the existing cryptographic hardware infrastructure on the Z mainframe to encrypt and decrypt the data using 256-bit AES data keys.

No changes to application programs are required in order to implement this feature. Currently, DFSMS encrypts sequential extended format data sets access through queued sequential access method (QSAM), basic sequential access method (BSAM), and Virtual Storage Access Method (VSAM) extended format data sets accessed through VSAM and VSAM record-level sharing (VSAM RLS):

- Key-sequenced data set (KSDS)
- Entry-sequenced data set (ESDS)
- Relative record data set (RRDS)
- Virtual relative record data set (VRRDS)
- Linear data set (LDS)

To create an encrypted data set, a secure AES data key is assigned to it by associating the AES key’s key label with the data set when it is allocated. For each encrypted data set, its key label is stored in the catalog.

The cryptographic key is in a secure repository managed by the Integrated Cryptographic Service Facility (ICSF), known as the *Cryptographic Key Data Set* (CKDS). The CKDS contains cryptographic keys used by z/OS components and applications and the only way to access these keys is by their key labels. Access to each key should be restricted by an external security manager like RACF. When the data set is to be decrypted, DFSMS gets the key label from the catalog and passes it to ICSF.

If the user has access to this key, ICSF decrypts the data with the cryptographic hardware as illustrated in Figure 1-10.

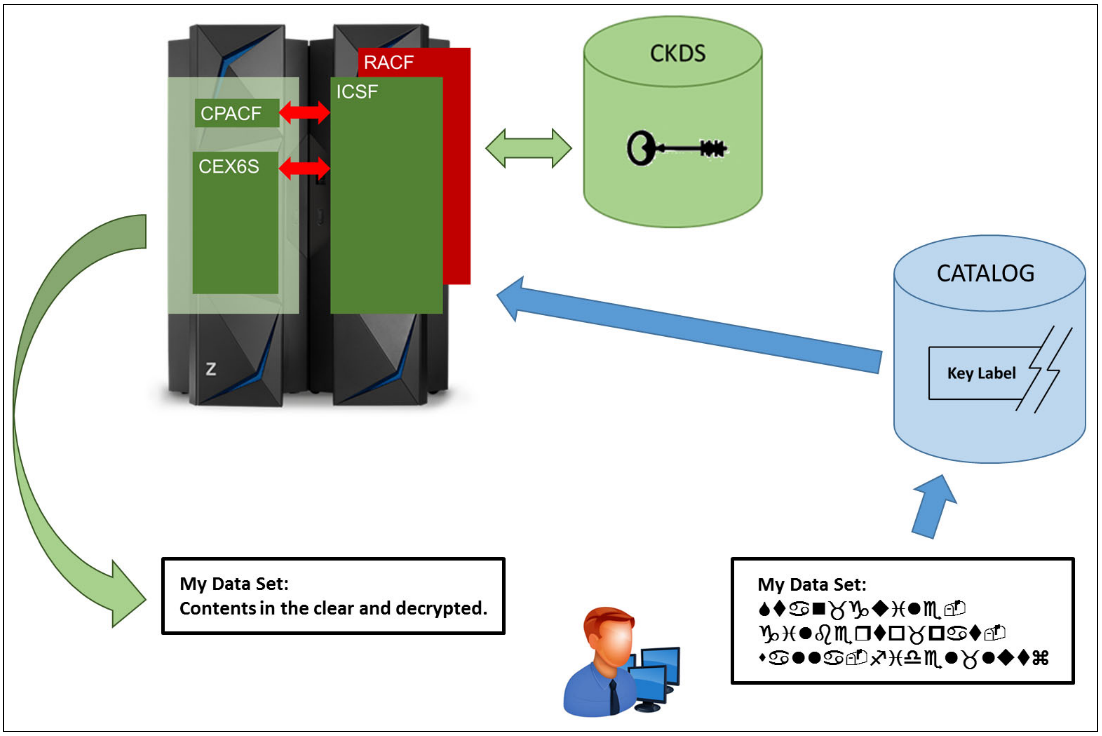

Figure 1-10 Decrypt data set

One of the benefits of data set encryption is that it eliminates the system and storage administrators from the compliance scope of audits. In order to read the contents of a data set, a person needs access to the cryptographic key that encrypted it. All the administrator needs to manage a data set is to have access to it and not the key. Without the key, the administrator will never be able to access the contents of data set.

Requirements

The minimum hardware requirements are Crypto Express3 and CPACF but the recommended hardware level is the cryptographic hardware available on the z14. The operating system requirement is z/OS V2.3 or z/OS 2.2 with the full function support service maintenance. Also, if the encrypted data set is moved to a different environment, the key used to encrypt the data needs to be in the CKDS of the new environment.

1.9 IBM Health Checker for z/OS

IBM Health Checker for z/OS is a z/OS base element. Its objective is to identify potential problems before they affect your availability or, in worst cases, cause outages. It checks the current active z/OS and sysplex settings and definitions for a system and compares the values to those suggested by IBM or defined by you. It is not meant to be a diagnostic or monitoring tool, but rather a continuously running preventive tool that finds potential problems.

IBM Health Checker for z/OS produces output in the form of detailed messages to notify you of both potential problems and suggested actions to take.

These messages do not mean that IBM Health Checker for z/OS has found problems that you need to report to IBM. Rather, IBM Health Checker for z/OS output messages inform you of potential problems so that you can expedite your installation.

Beginning at z/OS V2.3 or with OA49807 on z/OS V2.2, you can conditionally activate health checker policy statements for specific systems or sysplexes within the HZSPRMxx PARMLIB member. This feature eliminates the need for multiple, system-specific HZSPRMxx PARMLIB members. The ability to perform a syntax check on your HZSPRMxx PARMLIB members, without applying your changes to the existing environment, is also available.

1.9.1 Health Checker for z/OS processing

As illustrated in Figure 1-11, Health Checker for z/OS functions in the following way:

1. Check values provided by components.

   Each check includes a set of predefined values:

   – Interval, or how often the check runs
   – Severity of the check, which influences how check output is issued
   – Routing and descriptor codes for the check

   You can update or override certain check values using either the Spool Display and Search Facility (SDSF) or statements in the HZSPRMxx parmlib member, or by using the **MODIFY** command. For further information about SDSF, see Chapter 3, "TSO/E, ISPF, JCL, and SDSF" on page 27.

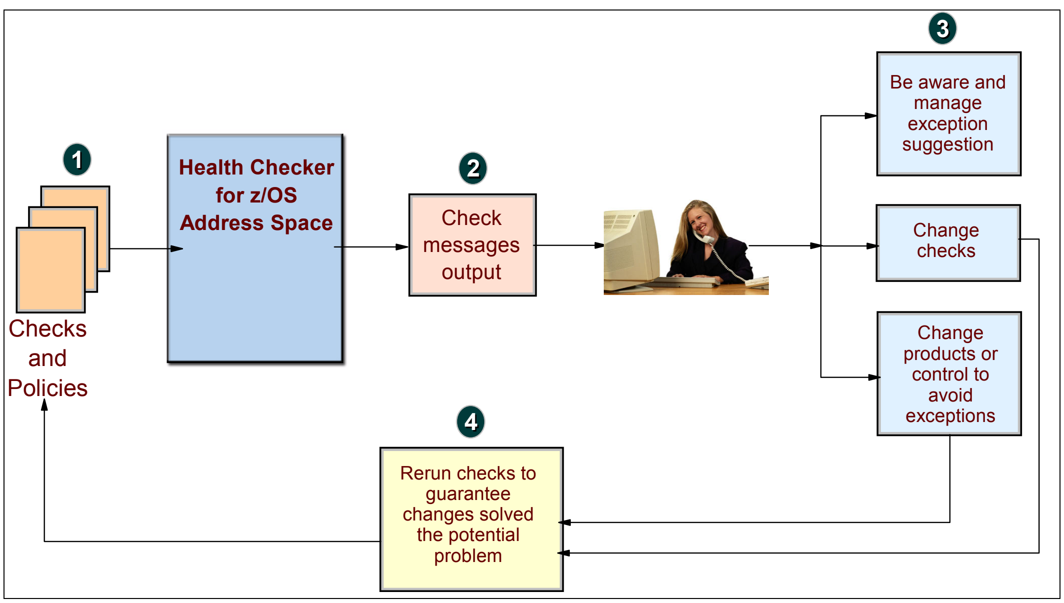

Figure 1-11 IBM Health Checker for z/OS

2. Check output.

   A check issues its output as write to operators (WTOs) and other messages, which you can view using IBM System Display and Search Facility (SDSF), the HZSPRINT utility, or a log stream that collects a history of check output.

   If a check finds a deviation from leading practices or a potential problem, it issues a WTO message known as an *exception*, as previously mentioned. Check exception messages include a description of the potential problem found, including the severity, but also information about what to do to fix the potential problem.

3. Resolve check exceptions.

   To get the best results from IBM Health Checker for z/OS, let it run continuously on your system so that you know when your system has changed dynamically from preferred practice values. When you get an exception, resolve it using the information in the check exception message, or by overriding check values, so that you do not receive the same exceptions over and over. You can use either SDSF or the `HZSPRMxx` parmlib member, or the IBM Health Checker for z/OS `MODIFY` (**`F hzsproc`**) command to manage checks.

4. If you solve an exception by changing a product setting or system control, it is a good policy to rerun the checks related to this action to ensure that the exception conditions have been resolved.

1.10 IBM z/OS Management Facility

The goal of the IBM z/OS Management Facility (z/OSMF) architecture is to provide simplified systems management functions through a common, easy-to-use GUI. Figure 1-12 shows a typical architecture and flow, starting with the user's browser session and continuing through z/OSMF and IBM WebSphere® Application Server Liberty Profile, with information passed to various z/OS system components as needed.

z/OSMF is a central system management function for z/OS and is designed to provide better tools for managing systems and helping system programmers to be more productive. z/OSMF provides a framework for managing various aspects of z/OS systems through an intuitive web 2.0 browser user interface and new enabling technologies on z/OS.

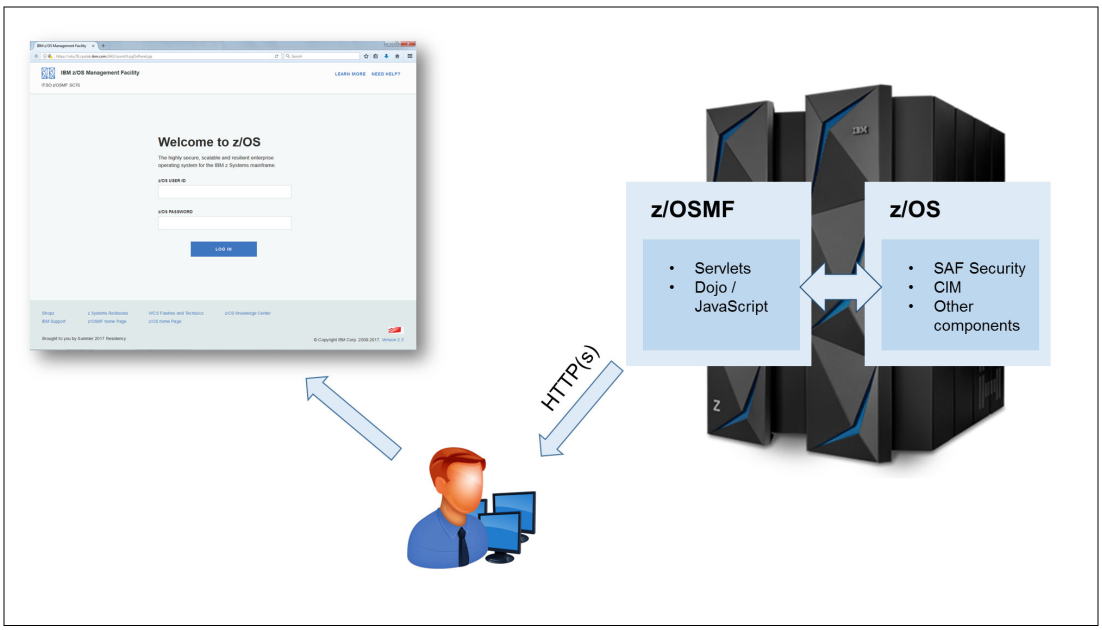

Figure 1-12 z/OSMF architecture

The z/OSMF application is written in Java so it can run on a specialized mainframe processor known as an IBM zEnterprise® Application Assist Processor (zAAP). Because the zAAP is not available on z13 and z14, the zAAP workload runs on a different specialized processor known as an IBM z® Integrated Information Processor (zIIP). The advantage of running any workload on a zIIP is that the workload does not incur any software license charges.

z/OSMF is now a required component of z/OS and is expected to be installed and configured on at least one system in every sysplex. z/OSMF became a base element of the operating system as of z/OS V2.2. Beginning with z/OS V2.3, z/OSMF starts automatically during an IPL.

The following is a list of z/OSMF tasks:

- Configuration Assistant
- Capacity Provisioning
- System Status and Resource Monitoring
- Workload Management
- Incident Log
- ISPF
- Workflow
- Software Management
- Sysplex Management
- Operator Consoles

For more information about z/OSMF, see *IBM z/OS Management Facility V2R1*, SG24-7851.

2

The z/OS system programmer

In a mainframe IT installation, system programmers are responsible for installing, customizing, and maintaining the z/OS operating system (OS). Their responsibilities also include installing or upgrading all additional products that run on the system, including middleware products. *Middleware* is a software layer between the OS and the user, or user application. Middleware supplies major functions that are not provided by the operating system.

This chapter covers the following topics:

- The role of the system programmer
- z/OS system programmer management overview
- The system programmer and z/OS operations
- Ordering z/OS

2.1 The role of the system programmer

The role of a system programmer is broad, but essentially it is to maintain a stable operating environment for users and business applications. A system programmer's responsibility can be at the operating system level, at the subsystem or middleware level, or at the hardware level. A system programmer installs, customizes, and maintains that environment by rolling out regular maintenance or even upgrading the entire environment to keep current and use new functions.

Another part of the system programmer's responsibility is to provide technical support to the users of the environment. This support can be answering questions about a product or analyzing a potential defect in the product (IBM or third party).

When a problem occurs in an address space, it sometimes abnormally ends or ABENDs. An address space is a virtual storage construct where code and control blocks are loaded in order for a started task, a job, or a TSO environment to run (see Chapter 5, "z/OS storage concepts" on page 67 for more details). Depending on how the recovery code is written, a data dump of the address space might be produced.

If you think there is a defect in the vendor code, a data dump and other diagnostic data (like JOBLOG) can be sent to the vendor for further investigation. If no dump is produced, you need to set a SLIP trap and re-create the problem to gather supporting documentation of the error. If there is a system-wide problem, then you need to perform a stand-alone data dump.

2.2 z/OS system programmer management overview

As a z/OS system programmer, you need to be involved in the customization of the items illustrated in Figure 2-1 on page 19. These items are explained in the following list:

- Address spaces. When z/OS is started, z/OS establishes system component address spaces. During the initial program load (IPL), the first address space started is the master scheduler address space (*MASTER*). There are other system address spaces for various subsystems and system components. See Chapter 4 for more details about address spaces.
- Paging. Page data sets contain the paged-out portions of address spaces, the common service area (CSA), pageable link pack area (PLPA), and the data written to virtual I/O (VIO) data sets.
- Dispatching work. The scheduling of address spaces as dispatchable units to run on a central processor (CP) in the z/OS system is done by the z/OS dispatcher component. The z/OS dispatcher is responsible for finding and dispatching the highest priority dispatchable unit in the system (Service Request Blocks or SRBs and tasks).
- Job flow. z/OS uses IBM Job Entry Subsystem (JES) to receive jobs (also called *batch*, which is a non-interactive type of transaction) into the OS, to schedule them for processing by z/OS, and to control their output processing. JES is the component of the OS that provides supplementary job management and data management.

  JES also provides supplementary task management functions, such as scheduling, control of job flow, and spooling (storing output on direct access storage device (DASD) spool volumes rather than printing them at the moment they are produced).

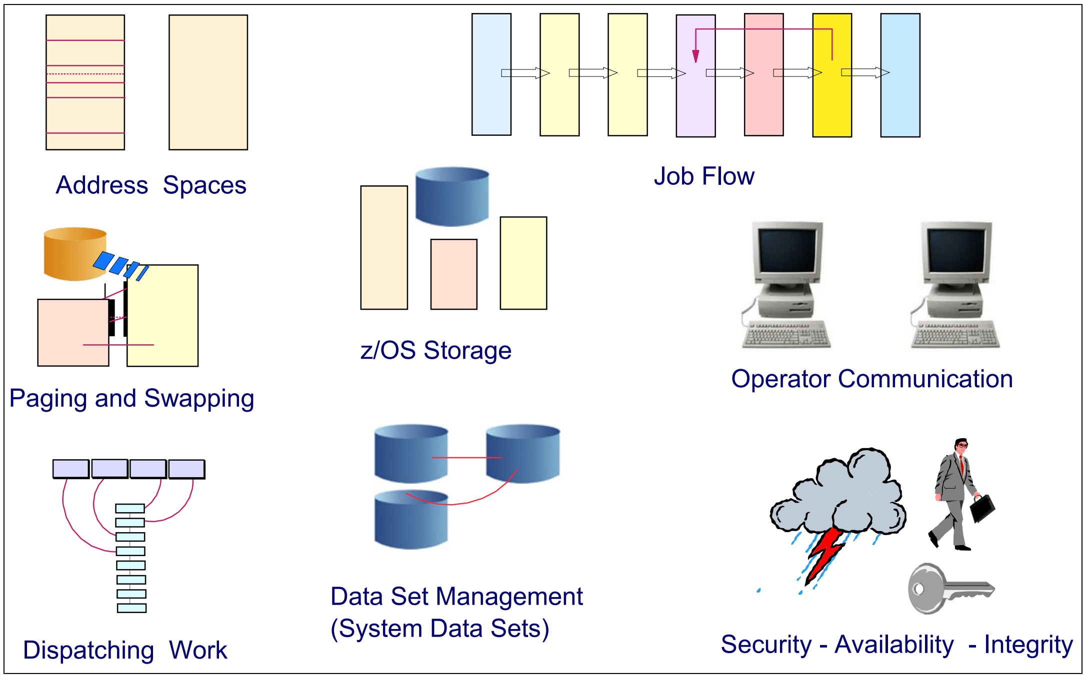

Figure 2-1 z/OS system programmer management overview

- z/OS storage. The system programmer must be aware of all storage considerations when installing and customizing a z/OS environment. The initialization process begins when the system operator selects the LOAD (IPL) function at the system console. z/OS locates all of the usable main storage that is online and available to the system, and creates a virtual environment for the building of various system areas.

  This initialization phase allocates the system's minimum virtual storage for the system queue area (SQA) and the extended SQA, allocates virtual storage for the extended local system queue area (extended LSQA) for the master scheduler address space, and allocates virtual storage for the CSA and the extended CSA. The amount of storage allocated depends on the values specified on the CSA system parameter read during the IPL.

- System data sets. Each installation must incorporate required system data sets into the system by allocating space for them on appropriate DASD during system installation. The DEFINE function of Access Method Services is used to define both the space requirements and the volume for each system data set. Some data sets must be allocated on the system residence volume (the volume that has the kernel of z/OS code). Other data sets can be placed on other direct access volumes.

- Operator communication. The operation of a z/OS system involves the following elements:

  – Console operations, or how operators and system programmers interact instantaneously with z/OS to monitor or control the hardware and software.

  – Message (produced by z/OS) processing and command (produced by an operator) processing that forms the basis of operator interaction with z/OS, and the basis of z/OS automation.

  – Managing hardware, such as processors and peripheral devices (including the consoles where operators or system programmers do their work), and software, such as the z/OS operating control system, JES, subsystems (such as IBM Tivoli® NetView® for z/OS) that can control automated operations, and all of the applications that run on z/OS.

- Security. Data security is the protection of data against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional. A security system, such as IBM Resource Access Control Facility (RACF), must be installed in your OS by a system programmer to maintain the resources necessary to meet the security objectives. The system programmer has the overall responsibility, using the technology available, to transform the objectives of the security policy into a usable plan.
- Availability. The software products supporting system programmers and operators in managing their systems heavily influence the complexity of their job and their ability to keep system availability at a high level. Performance management is the system management discipline that most directly impacts all users of system resources in an enterprise and can be achieved, for example, by using IBM Resource Measurement Facility™ (RMF).
- Integrity. An OS is said to have system integrity when it is designed, implemented, and maintained to protect itself against unauthorized access, and does so to the extent that security controls specified for that system cannot be compromised. Specifically for z/OS, there must be no way for any unauthorized program, using any system interface, defined or undefined, to perform the following actions:
  - Bypass store or fetch protection.
  - Bypass password use, Virtual Storage Access Method (VSAM) password, or RACF security checking.
  - Obtain control in an authorized state.

For additional reference, see *ABCs of z/OS System Programming Volume 2*, SG24-6982.

2.3 The system programmer and z/OS operations

A system programmer is involved in the planning of daily system operations in various areas (see Figure 2-2).

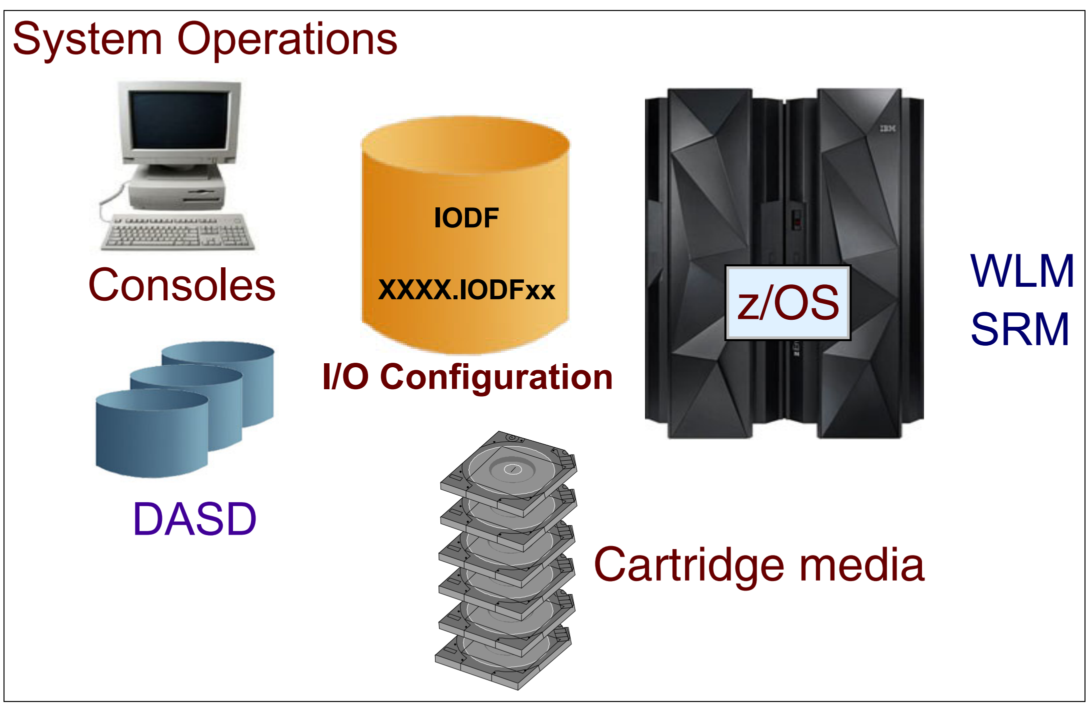

Figure 2-2 The system programmer and z/OS operations

2.3.1 Planning

A system programmer has to plan the following operations areas:

- z/OS Workload Manager

  z/OS Workload Manager (WLM) provides a solution for managing workload distribution, workload balancing, and distributing resources to competing workloads. Managing workloads is possible due to the combined cooperation of various subsystems (CICS, IBM IMS/ESA®, JES, Advanced Program-to-Program Communication (APPC), Time Sharing Option Extensions (TSO/E), z/OS UNIX System Services, distributed data facility (DDF), IBM DB2®, System Object Model (SOM), LAN Server for MVS (LSFM), and Internet Connection Server) with the WLM component.

- System performance

  The task of tuning a system is an iterative and continuous process. The controls offered by system resources manager (SRM) are only one aspect of this process. Initial tuning consists of selecting appropriate parameters for various system components and subsystems. After the system is operational and criteria have been established for the selection by job classes and priorities of jobs to run, SRM will control the distribution of available resources according to the parameters specified by the installation.

  However, WLM can only deal with available resources. If these are inadequate to meet the needs of the installation, even optimal distribution might not be the answer. Other areas of the system should be examined to determine the possibility of increasing available resources.

  When requirements for the system increase and it becomes necessary to shift priorities or acquire additional resources, such as a larger processor, more storage, or more terminals, the WLM goals might have to be adjusted to reflect changed conditions.

- I/O device configuration

  As a system programmer, you must define an I/O configuration to the operating system (software) and the channel subsystem (hardware). The Hardware Configuration Definition (HCD) component of z/OS consolidates the hardware and software I/O configuration processes under a single interactive user interface. The validation checking that HCD does as you enter data helps to eliminate errors before you attempt to use the I/O configuration.

  The output of HCD is an IODF, which contains the server, the logical partitions and the I/O configuration data. An IODF is used to define multiple hardware (servers) and software configurations to z/OS. When you activate an IODF, HCD defines the I/O configuration to the channel subsystem or the operating system.

  With the HCD activate function or the z/OS `ACTIVATE` operator command, you can change the current configuration without having to restart the software or perform a Power-on Reset (POR) of the hardware. Making changes while the system is running is known as *dynamic configuration* or *dynamic reconfiguration*.

- Console operations

  The operation of a z/OS system involves the following elements:

  – Console operations, or how operators interact with z/OS to monitor or control the hardware and software

  – Message and command processing that forms the basis of operator interaction with z/OS and the basis of z/OS automation

  Operating z/OS involves managing hardware, such as processors and peripheral devices (including the consoles where your operators do their work).

You also manage software, such as the z/OS operating control system, the job entry subsystem, subsystems (such as IBM Tivoli NetView for z/OS) that can control automated operations, and all of the applications that run on z/OS.

Planning z/OS operations for a system must consider how operators use consoles to accomplish their work and how you want to manage messages and commands. Because messages are also the basis of automated operations, understanding message processing in an z/OS system can help you plan z/OS automation.

- z/OS Installation

  Plan and optimize the z/OS installation in order to make system upgrades and maintenance easier to perform.

2.3.2 Daily operations

Also involved are the business goals and policies established to enable the installation to grow and handle work efficiently. These needs, of course, vary from installation to installation, but they are important when you plan your z/OS operations.

Managing the complexity of z/OS requires you to think about the particular needs of your installation. However, installations can consider the following goals when planning z/OS operations.

- Increasing system availability

  Many installations need to ensure that their system and its services are available and operating to meet service level agreements (SLAs). Installations with 24-hour, 7-day operations need to plan for minimal disruption of their operation activities. In terms of z/OS operations, how the installation establishes console recovery or whether an operator must restart a system to change processing options are important planning considerations.

- Controlling operating activities and functions

  As more installations make use of multisystem environments (as in IBM Parallel Sysplex), the need to coordinate the operating activities of those systems becomes crucial. Even for single z/OS systems, you must consider controlling communication between functional areas (such as a tape-pool library and the master console area, for example).

  In both single and multisystem environments, the commands that operators can issue from consoles can be a security concern that requires careful coordination. As a planner, ensure that the correct people are performing the correct tasks when they interact with z/OS. If your installation uses remote operations to control target systems, you also need to consider how to control those activities from the host system.

- Simplifying operator tasks

  Because the complexity of operating z/OS has increased, the tasks and skills of operators also require careful consideration. How operators respond to messages at their consoles and how you can reduce or simplify their actions are important to operations planning. Further, planning z/OS operator tasks in relation to any automated operations that help simplify those tasks is also needed.

- Streamlining message flow and command processing

  In thinking about operator tasks, consider how to manage messages and commands. Operators need to respond to messages. Routing messages to operator consoles, suppressing messages to help your operators manage increased message traffic, and selecting messages for automated operations can all help you manage system activity efficiently.

- Single system image

  Single system image enables the operator, for certain tasks, to interact with several images of a product as though they were one image. For example, the operator can issue a single command to all z/OS systems in the sysplex rather than repeating the command for each system.
- Single point of control

  Single point of control enables the operator to interact with a suite of products from a single workstation. An operator can accomplish a set of tasks from a single workstation, thereby reducing the number of consoles that the operator has to manage.

2.4 Ordering z/OS

z/OS and other IBM zSeries software products are ordered through the IBM internet application called Shopz at https://www.ibm.com/software/shopzseries (see Figure 2-3). Using Shopz, you can also order corrective and preventive service.

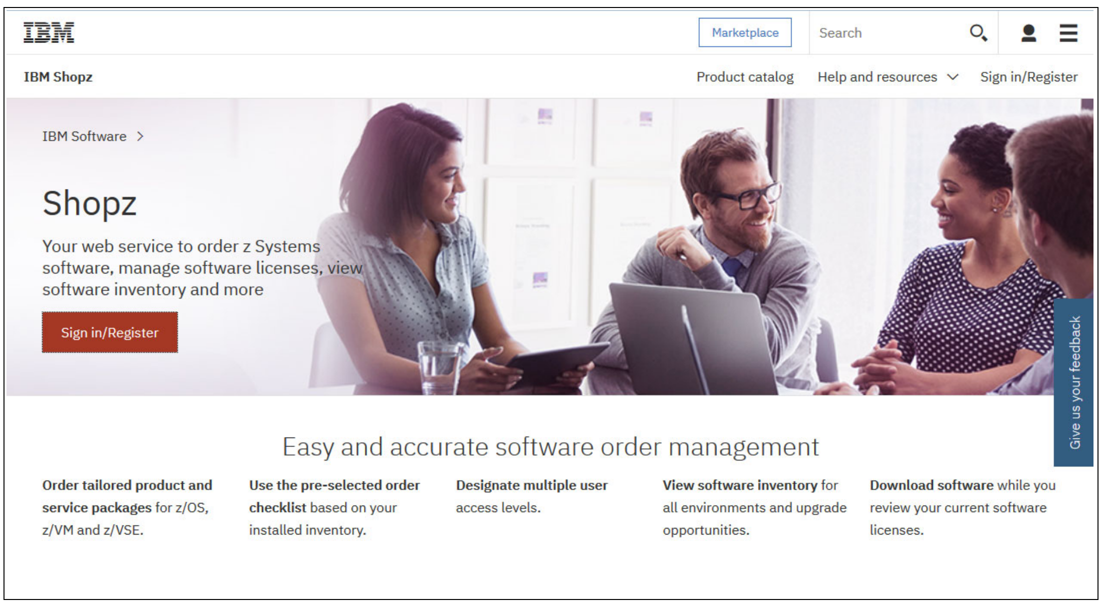

Figure 2-3 Shopz home page c.2017

You need to register to get access to Shopz. Product entitlement is based on your customer number. After you log in, you are in the My Shopz page (see Figure 2-4) where you can order products, corrective service, and preventive service.

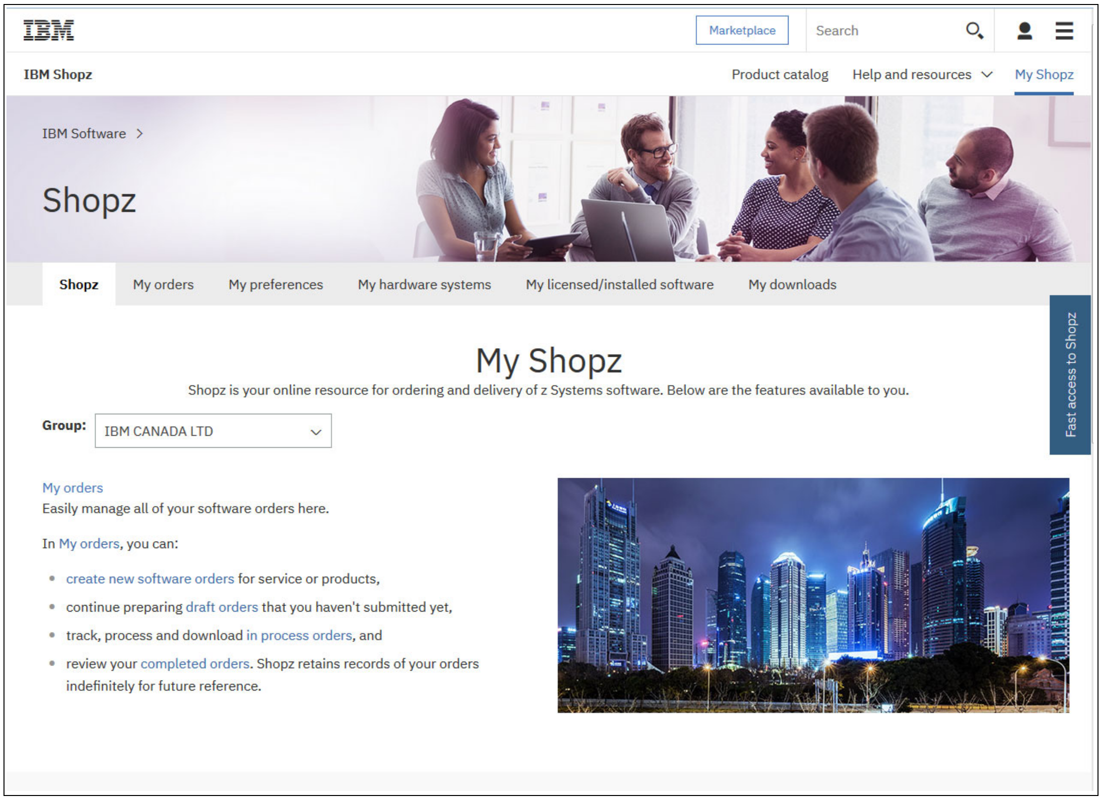

Figure 2-4 My Shopz page c. 2017

You can receive your order on physical media, such as IBM 3590 and 3592 tape or DVD, or you can download your order from the internet. IBM intends to discontinue delivering products and maintenance on tape in the future so the preferred delivery method is internet download.[1]

2.4.1 z/OS delivery options

Several IBM packages are available for installing z/OS. Some packages are entitled with the product (as part of your z/OS license, at no additional charge). Other packages are available for an additional fee. This section describes each package:

ServerPac ServerPac is an entitled software delivery package consisting of products and service for which IBM has performed the SMP/E (System Modification Program/Extended) installation steps and some of the post-SMP/E installation steps. To install the package on your system and complete the installation of the software that it includes, you use the CustomPac Installation dialog. For ServerPac orders, service is integrated with product code.

[1] Preview: IBM z/OS Version 2 Release 3 - Engine for digital transformation (Software Announcement A17-0134)

CBPDO CBPDO is an entitled software delivery package consisting of uninstalled products and unintegrated service. There is no dialog program to help you install it, as there is with ServerPac. You must use SMP/E to install the individual z/OS elements and features, and their service, before you can use it. Installation instructions are in the z/OS Program Directory.

SystemPac SystemPac is a software package, available for an additional fee and offered worldwide, that helps you install z/OS, subsystems (DB2, IMS, CICS, NCP, and WebSphere Application Server). SystemPac is tailored to your specifications; it is manufactured according to parameters and IODF definitions that you supply during order entry. The goal is to have the system tailored to your specifications and have products enabled according to your specified configuration.

2.4.2 SMP/E Internet Service Retrieval

Internet Service Retrieval (ISR) uses the SMP/E RECEIVE ORDER command to order software service directly from IBM via a secure download (https or ftps). There are two IBM Automated Delivery Request servers to choose from and their URLs are:

– `https://eccgw01.boulder.ibm.com/services/projects/ecc/ws`
– `https://eccgw02.rochester.ibm.com/services/projects/ecc/ws`

The service order will contain the latest program temporary fixes (PTFs) and HOLDDATA for your system, and you can download it when the order is fulfilled.

With SMP/E ISR, you can request service on demand and even automate the service delivery process. For example, you can schedule a SMP/E job to run once a week or even every night to order and download the latest HOLDDATA and critical PTF service. It might not be practical to run the job every night because the regular maintenance window is the first Sunday of every month. For reference, see the following website:

`http://www14.software.ibm.com/webapp/set2/sas/f/gdbm/home.html#outages`

3

TSO/E, ISPF, JCL, and SDSF

This chapter describes the basic products that a system programmer needs to install and customize an IBM z/OS operating system:

- Time Sharing Option/Extensions (TSO/E)

  TSO/E allows users to create an interactive session with the z/OS. TSO/E provides a single-user logon capability and a basic command-line interface (CLI) to z/OS.

- Information Center Facility

  The Information Center Facility is the foundation for building an z/OS-based information center (IC). An IC increases user productivity and the computer effectiveness by providing easy-to-use computing tools, data access, education, and other assistance for users who have little or no data processing experience.

- Interactive System Productivity Facility (ISPF)

  ISPF is a full-panel application navigated by a keyboard. ISPF includes a text editor and browser, and functions for locating and listing files and performing other utility functions.

  After logging on to TSO, users typically access the ISPF menu. In fact, many use ISPF exclusively for performing work on z/OS. ISPF menus list the functions that are most frequently needed by online users.

- TSO/E and ISPF are used to perform the following tasks:
  - Install and customize z/OS and other products
  - Communicate interactively with the operating system
  - Define and maintain user definitions
  - Create data sets and JCL, and submit jobs
  - Communicate with other TSO/E users
  - Develop and maintain programs in languages, such as assembler language, Common Business Oriented Language (COBOL), Fortran, Pascal, C, C++, Java, PL/I, Restructured Extended Executor (REXX), command list (CLIST), and so on
  - Manipulate data

- Job control language (JCL)

  Job control language (JCL) is a set of statements that you code to tell the z/OS operating system about the work you want it to perform. JCL statements tell z/OS where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output.

- System Display and Search Facility (SDSF)

  SDSF provides a powerful and secure way to monitor and manage your z/OS system, in both JES2 and JES3 environments. SDSF's easy-to-use interface lets you control the following elements:

  - Jobs and output
  - Devices, such as printers, readers, lines, and spool offloaders
  - Checks from IBM Health Checker for z/OS
  - System resources, such as WLM scheduling environments, the members of your Multi Access Spool (MAS), and JES job classes
  - System log and action messages

  SDSF function is available in IBM z/OS Management Facility (z/OSMF), which lets you manage aspects of z/OS through a web browser interface. You can see system activity at a glance with graphic views of CPU use; manage and browse jobs, output, and checks for IBM Health Checker for z/OS; issue system commands and export SDSF tables for use by other programs, such as spreadsheet applications.

  This chapter covers the following topics:

- TSO/E
- Interactive System Productivity Facility
- Job control language
- System Display and Search Facility

3.1 TSO/E

Time Sharing Option/Extensions (TSO/E) allows users to create an interactive session with the z/OS system. LOGON is the TSO/E command that tells TSO/E you want to begin a computer session. TSO/E provides a single-user logon capability and a basic command prompt interface to z/OS. For more information about TSO/E, see *z/OS TSO/E Primer*, SA32-0984 and *z/OS TSO/E General Information*, SA32-0979 manuals.

Most users work with TSO/E through its menu-driven interface, Interactive System Productivity Facility (ISPF). This collection of menus and panels offers a wide range of functions to assist users in working with data files on the system. ISPF users include system programmers, application programmers, administrators, and others who access z/OS. In general, TSO/E and ISPF make it easier for people with varying levels of experience to interact with the z/OS system.

In a z/OS system, each user is granted a user ID and a password authorized for TSO/E logon. The TSO/E user ID is a group of characters that identifies the user. It is unique to the installation; it can be no longer than eight (8) characters and can contain numeric (0 - 9) and alphabetic (A to Z) characters, but must begin with an alphabetic character. Logging on to TSO/E requires a 3270 display device or, more commonly, a TN3270 emulator running on a PC.

During TSO/E logon, the system displays the TSO/E logon panel on the user's 3270 display device or TN3270 emulator.

Figure 3-1 shows a typical example of a TSO/E logon panel.

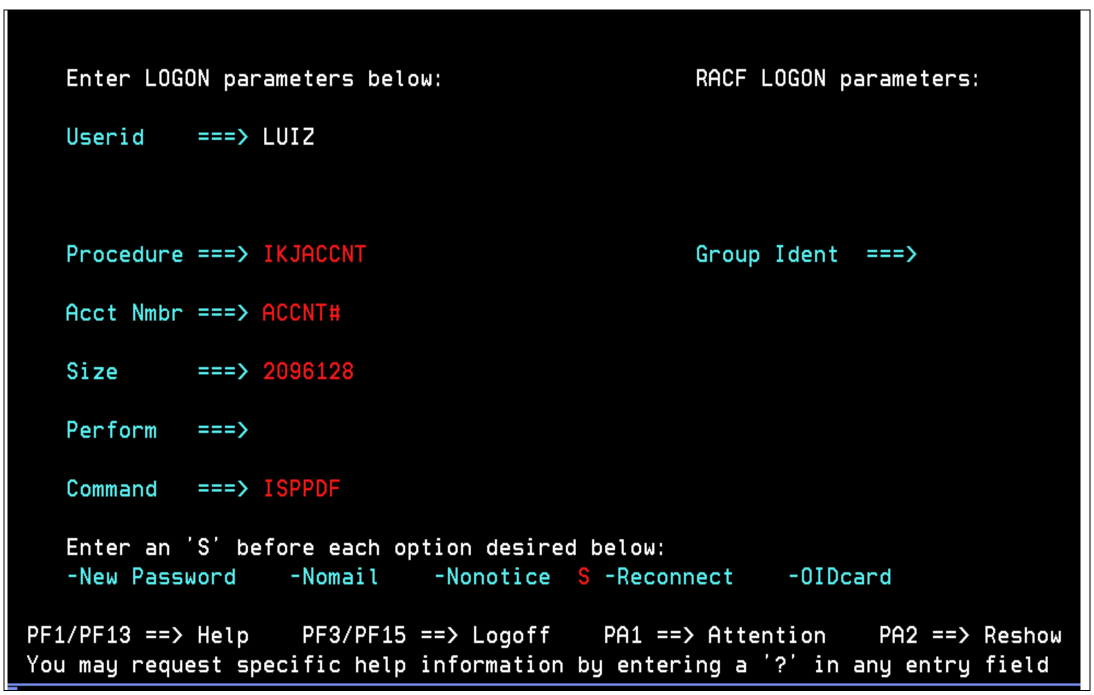

Figure 3-1 TSO/E Logon Panel

Many of the panels that are used with ISPF show program function (PF) key settings. Because it is common practice for z/OS sites to customize the PF key assignments to suit their needs, the key assignments shown in Figure 3-1 might not match the PF key settings in use at your site.

TSO/E is a base element of z/OS. TSO/E enables users to interactively share computer time and resources. TSO/E is the primary user interface to the z/OS operating system.

TSO/E provides programming services that you can use in system or application programs. These services consist of programs, macros, commands, and command lists (CLISTs). With TSO/E, it is possible to place a list of commands, CLIST, in a file, and run the list as though it were one command. When you invoke a CLIST, it issues the TSO/E commands in sequence. CLISTs are used for performing routine tasks; they enable users to work more efficiently with TSO/E. TSO/E services support a wide range of functions that are useful in writing both system programs and application programs that use the full-screen capabilities of TSO/E.

You can use TSO/E in any one of the following three environments:

- The Information Center Facility

  The Information Center Facility is the foundation for building an z/OS-based information center (IC). An IC increases user productivity and the computer effectiveness by providing easy-to-use computing tools, data access, education, and other assistance for users who have little or no data processing experience.

  The Information Center Facility eases users into the data processing environment by providing a series of conversational panels. These panels eliminate numerous command-driven interactions between the user and the system.

  In addition to user services, the Information Center Facility provides panels that enable an administrator to maintain the facility, enroll users, and add, modify, and delete products. TSO/E provides several new functions that help an administrator to easily install, maintain, and upgrade products in the Information Center Facility.

  TSO/E also provides support for tailoring Information Center Facility panels, functions, and environments to the needs of different departments or user groups and individual users. This enhancement eliminates the requirement to make all products in the Information Center Facility available to all users.

  This is easiest way to use TSO/E; it provides a way to display many services, such as:

  – Office support – (mail, names directory)
  – Decision support – (data analysis)
  – Document preparation – (reports, charts, graphs)

- ISPF

  The Interactive System Productivity Facility (ISPF) works together with TSO/E to provide panels with which users can interact. ISPF provides the underlying dialog management service that displays panels and enables a user to navigate through the panels. Program Development Facility (PDF) is a dialog of ISPF that helps maintain libraries of information in TSO/E and enables a user to manage the library through different facilities, such as browse, edit, and utilities.

- Line mode TSO/E

  This is the way programmers originally communicated interactively with the z/OS operating system; it uses TSO/E commands typed on a terminal, one line at a time. It is a quick and direct way to use TSO/E.

TSO/E is a powerful tool for both programmers and non-programmers. The introduction of new products and the use of Information Center Facility and ISPF is moving TSO/E from a system programmer’s tool to a highly flexible subsystem that delivers computing resources to the user. Anyone can use TSO/E to do many diverse tasks in several environments.

CLISTs, REXX execs (see “Restructured Extended Executor (REXX) language support” on page 32), servers, and command processors are specific types of programs that you can write to run in the TSO/E environment.

TSO/E offers advantages to a wide range of computer users, including system programmers, application programmers, information center administrators, information center users, TSO/E administrators, and others who access applications that run under TSO/E.

3.1.1 TSO/E highlights

Highlights of TSO/E are described in this section.

Session Manager

The TSO/E Session Manager is an interface to line mode TSO/E. TSO/E Session Manager keeps a complete journal of everything that happens during your terminal session while you are in line mode TSO/E. It records everything you type in and everything the system displays.

Any time during your terminal session, you can look at work you did in the beginning, middle, or end of your session. TSO/E Session Manager also lets you print a copy of this information. You can correct or change a command that is displayed on the window without having to retype the entire command. By enabling you to redisplay, change, and reuse your input, the Session Manager makes TSO/E easier to use.

Commands

TSO/E provides numerous commands for both end users and programmers that allow them to interact with TSO/E and the z/OS system. The **ALLOCATE**, **FREE**, and **EDIT** commands are examples of commands that allow users to manage their data sets. The **CONSOLE** command enables users with **CONSOLE** command authority to perform z/OS operator activities from a TSO/E session. It is important to notice that the use of these commands must be authorized by the installation.

Online help

Terminal users can obtain online help for most TSO/E commands. Information Center Facility users can obtain help for each panel and message. The HELP facility provides installations greater flexibility in adding help information. Installations can also provide help information in different languages.

Data and notice handling

TSO/E simplifies how data and notices are sent and received. For example, the **TRANSMIT** and **RECEIVE** commands enable users to send data and messages to other users in a network. The broadcast data set or individual user logs contain messages that either the system or another user sends using the **SEND** command.

Logon processing

TSO/E provides a full-screen logon panel that makes the logon process easier:

- Saving user attributes from one session to the next
- Enabling program function keys to be used during logon
- Enabling users to enter commands during logon
- Explaining the error when incorrect information is specified

Users can request private areas of up to 2 gigabytes for each terminal session. Your installation can also customize the logon panel and the logon help panel, and customize logon processing (see “TSO/E logon process in a VTAM environment” on page 34).

Language enablement

TSO/E enables installations to provide TSO/E messages and the TRANSMIT full-screen panel to users in different languages. The TSO/E CONSOLE command also supports the display of translated system messages issued during a console session.

Help information can be provided in a language defined by the installation.

Support for logon panels and their help text in different languages is also available.

The TSO/E REXX supports arguments that REXX execs can use to obtain language information. Execs can use this information together with the **SETLANG** function to set the language in which REXX messages are displayed.

Security

Installations installed can use security labels (SECLABELs) to protect system resources. TSO/E provides support to use security labels. The **LOGON** command and full-screen logon panel support the specification of a security label to be associated with a user's TSO/E session. The **SUBMIT** command enables users to submit jobs at a security label that is greater than the one they are currently logged on with. You can use the **OUTDES** command enhancements to print the job's security label on each page of output.

Installations can also control communication between users to protect the security classification of information. For example, installations can control and audit the use of the **SEND** command. **LISTBC** command processing enables installations to restrict users from viewing messages for which they do not have the proper security.

CLIST language

The CLIST language is a high-level programming language that enables programmers to issue lists of TSO/E commands and JCL statements in combination with logical, arithmetic, and string-handling functions provided by the language. CLISTs can simplify routine user tasks, start programs written in other languages, and perform complex programming functions.

Restructured Extended Executor (REXX) language support

TSO/E provides REXX support to z/OS users. REXX is a high-level procedures language that enables inexperienced users and experienced programmers to write structured programs called REXX execs. You can run REXX execs in any z/OS address space (both TSO/E and non-TSO/E).

REXX language allows programmers to perform logical and arithmetic operations and communicate with terminal users. REXX built- in functions increase usability by allowing you to easily perform character manipulation and conversion operations.

TSO/E extends the programming capabilities of REXX by providing TSO/E functions, REXX commands, and programming and customizing services. These facilities allow you to perform additional tasks, such as controlling the execution of a REXX exec and customizing how system services are accessed and used. Some of these facilities are available only to REXX execs that run in the TSO/E address space.

REXX execs perform functions similar to CLISTs and can call, and be called by, existing CLISTs and other TSO/E programs. Therefore, REXX is an attractive alternative to the CLIST language.

TSO/E service facility

The TSO/E service facility enables TSO/E users to run authorized or unauthorized programs, TSO/E commands, or CLISTs from an unauthorized environment, while maintaining system integrity.

TSO Command Package

The TSO Command Package provides functions that help to improve productivity:

- Support for running terminal sessions as batch jobs
- Automatic saving of data
- Accounting facility
- Defaults for the user-attribute data set

Enhanced Connectivity Facility

The Enhanced Connectivity Facility (ECF) enables you to customize how host server programs and personal computer (PC) requester programs communicate. IBM products or customer-written programs can supply the services.

The user can access z/OS host services from a PC using IBM System/370-to-IBM Personal Computer ECF. This enables PC users to interact with z/OS or IBM Virtual Machine/System Product (VM/SP) systems using PC commands.

Support for z/OS UNIX

Installations can use the functions provided by the TSO/E `ALLOCATE` and `FREE` commands to manipulate z/OS UNIX files.

TSO/E Health Checks

TSO/E provides health checks that are registered automatically during system initialization. These checks run at regular intervals, and they can be disabled at any time. They alert installations to potentially serious problems so that corrective action can be taken to limit the impact.

3.1.2 TSO/E customization

TSO/E customization is the process whereby you tailor TSO/E functions to fit the specific needs of your installation. To tailor a TSO/E function, you might make a change or addition to TSO/E itself or to some other related IBM product or z/OS element, such as the Advanced Communications Function for Virtual Telecommunications Access Method (VTAM). For more information, see *z/OS TSO/E Customization*, SA32-0976.

Some customization is required before you can use TSO/E, but most is optional. If you elect not to do optional customization on a specific TSO/E function, that function then works according to IBM-provided defaults.

Each of the following z/OS elements and products interacts with TSO/E. They provide interfaces that you can use to customize the way TSO/E and the element work together. The elements are:

- DFSMSdfp
- ISPF
- Job Entry Subsystems (JES2 and JES3)
- Security Server (RACF)
- VTAM
- IBM Print Services Facility™ (PSF) for z/OS

Before users can use TSO/E, you must do several customization tasks. After you complete required customization, users will be able to log on and issue commands:

1. Define TSO/E to VTAM
2. Define those users who will be allowed to log on to TSO/E
3. Create at least one TSO/E logon procedure for users (see the following TSO/E logon procedure). You must define TSO/E address spaces to VTAM in order to use TSO/E; the VTAM APPL definition statements are used for this definition. You must code one APPL definition statement to define the primary TSO/E address space called TCAS, and at least as many APPL definition statements as there will be users logged on to TSO/E at one time.

   For example, if you want to allow 50 users to use TSO/E simultaneously, code 51 APPL definition statements: one for the primary TSO/E address space and one for each user address space.

Each user is defined to TSO/E by storing its user ID, logon procedure name, and the TSO/E resources that it has authority to use. This can be done, for example, by adding a user to one of the following data repositories:

- User Attribute data set (UADS), or
- IBM Resource Access Control Facility (RACF) database

When RACF is installed, it can be used to control access to the system and store information about each TSO/E user. The RACF database contains profiles for every entity (user, data set, or group) defined to RACF. For more information about RACF, see *z/OS Security Server RACF System Programmer's Guide*, SA23-2287.

3.1.3 TSO/E logon procedure

A TSO/E logon procedure contains JCL statements that run the required program and allocate the required data sets to enable a user to acquire the resources needed to use TSO/E. To log on to TSO/E, a user must have access to at least one logon procedure. The logon procedure is usually in a data set called `SYS1.PROCLIB`. The name of the procedure provided by TSO/E is `IKJACCNT` and is intended for system programmers to initially access the system.

TSO/E logon process in a VTAM environment

In a VTAM environment, when a user enters a **`LOGON`** command to TSO the following actions occur:

1. VTAM receives the command and passes it to the TCAS address space (the primary TSO address space).
2. If the maximum number of users logged on in the system is reached, the logon is rejected; if not, and the user ID was not specified, TCAS prompts for the user ID.
3. After the user ID is specified, TCAS verifies that the user has authority to use TSO/E. Depending on the installation customization, a full-screen logon panel is shown to the user. Figure 3-1 on page 29 shows the panel displayed when the user is RACF defined.

   The values shown in the fields PROCEDURE, ACCT NMBR, SIZE, and COMMAND are the same the user entered for the previous TSO/E session. If this is the first session, they are the default values.
4. After the Enter key is pressed, TSO/E verifies the values that were entered, then the user ID and the logon procedure name is passed to JES. The user address space is created (Master AS) and the required resources are allocated.

5. The user receives a window with the READY prompt at the left top corner of the window. This is called *line-mode TSO/E*. Now TSO/E is ready to accept commands, and user interfaces, such as ISPF or SDSF, can be called, as shown in Figure 3-2.

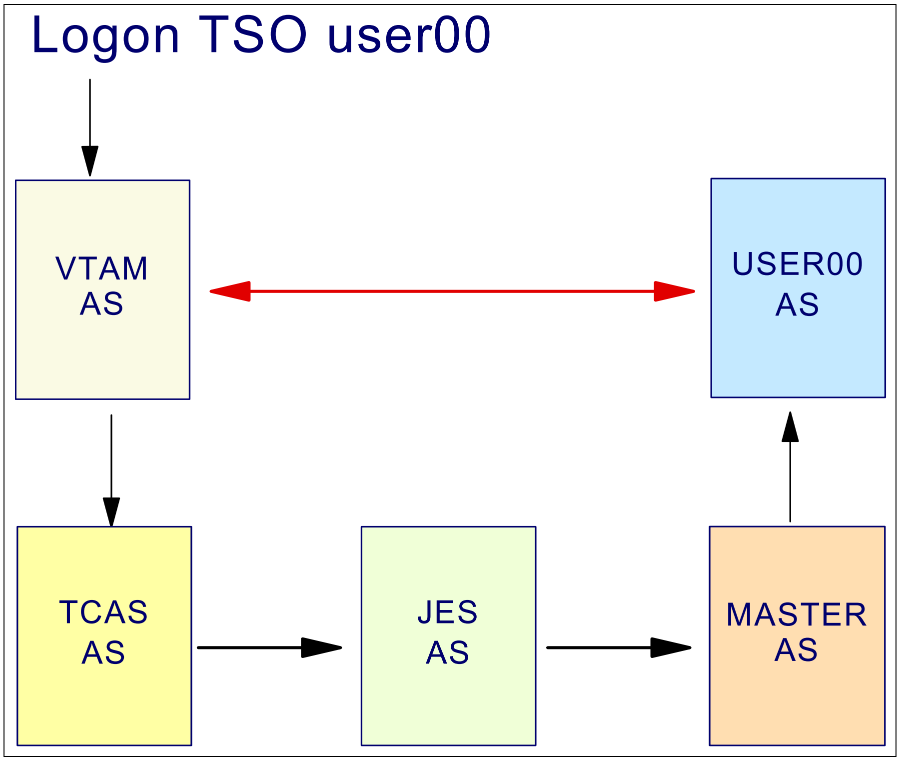

Figure 3-2 TSO/E logon process in a VTAM environment

3.1.4 Line mode TSO/E

When you do not enter a command name in the panel shown in Figure 3-1 on page 29, you enter line mode TSO/E. When you log on, you see the panel shown in Figure 3-3. The word `READY` in the corner indicates that TSO is ready to accept your commands.

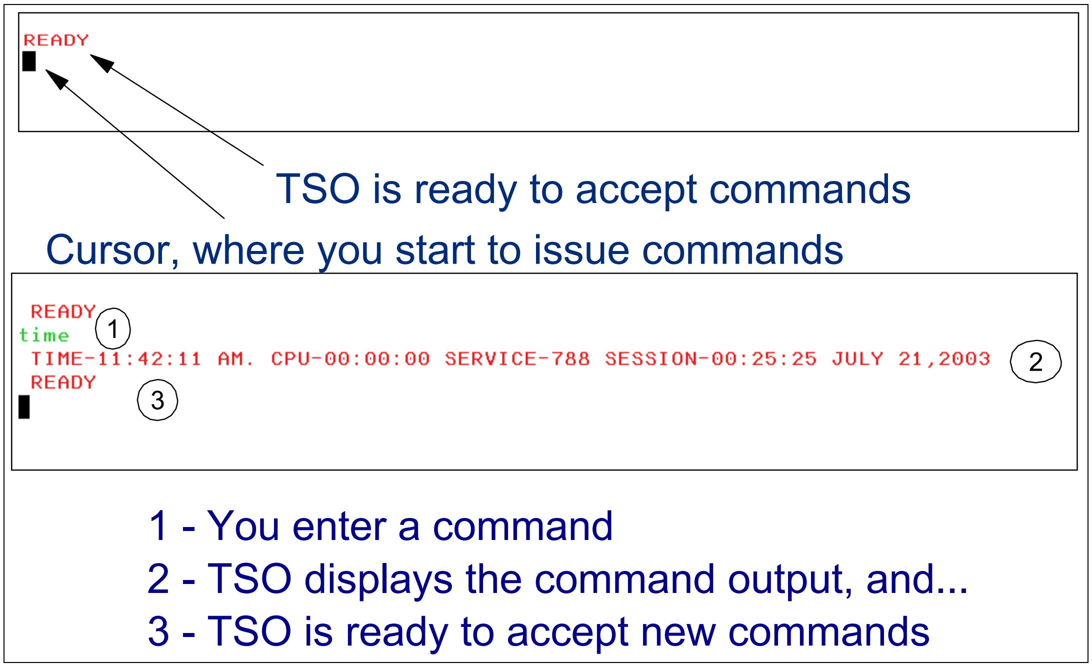

Figure 3-3 Line mode TSO/E

In line mode TSO/E, you type TSO/E commands one line at a time. It is a direct way to use TSO/E, and was the way programmers originally used to communicate interactively with the z/OS operating system.

You probably will not use line mode TSO/E. The user interface provided by ISPF is a more friendly way to work with TSO/E. For more information about using TSO/E and ISPF, see 3.2.1, "Using ISPF" on page 38.

For more information, see *z/OS TSO/E Primer*, SA32-0984, and *z/OS TSO/E User's Guide*, SA32-0971.

Interrupting a TSO/E function

The Attention Interrupt key enables you to interrupt or end a process that is taking place. If you are in a process and you want to stop, or you see a message requesting information that you do not have, you can press the Attention Interrupt key to end the process.

The Attention Interrupt key often is labeled `PA1`. Sometimes it is called an *escape* key and is labeled `Esc`.

You can end a program or a process by pressing the attention interrupt key. For instance, if you were running a program and the program went into a loop, you can press the attention interrupt key to stop processing.

3.1.5 TSO/E languages

There are two languages available in the TSO/E environment: REXX and CLIST. REXX and CLIST can be used to customize and tailor your TSO/E environment specifically for the applications you want to use. Figure 3-4 shows a REXX exec and a sample CLIST procedure.

❑ REXX

```
/*REXX     */
parse upper arg ax
Address 'ISPEXEC'
  "LIBDEF ISPPLIB DATASET ID('VAINI.U.PANELS')"
if rc = 0 then do
  Address "TSO"
  aplst = "'VAINI.U.CLIST'"
  if aplst ¬= "" then ,
    "ALTLIB ACT APPL(CLIST) DS("aplst")"
  Address "ISPEXEC"
    "SELECT CMD(DR" ax ") NEWAPPL(DREQ) PASSLIB "
  Address "TSO"
  if aplst ¬= "" then ,
    "ALTLIB DEACT APPL(CLIST)"
  Address 'ISPEXEC'
    "LIBDEF ISPPLIB "
end
else say '*** ERROR*** DR must be run under ISPF'
Exit 0
```

❑ CLIST

```
PROC 0 DB
IF .&DB = .DB THEN +
  CONTROL LIST SYMLIST CONLIST MSG PROMPT
CONCATD F(SYSPROC) DA('DB2V710S.NEW.SDSNCLST') BEFORE
CONCATD F(ISPLLIB) DA('DSN710.QPP.RUNLIB.LOAD') BEFORE
ISPEXEC SELECT CMD(DSNECPRI SSID(DB2S)) NEWAPPL(DSNE)
DECON F(ISPLLIB) DA('DSN710.QPP.RUNLIB.LOAD')
DECON F(SYSPROC) DA('DB2V710S.NEW.SDSNCLST')
```

Figure 3-4 TSO/E languages

REXX

The REstructured eXtended eXecutor (REXX) language is a high-level procedures language that enables inexperienced users and experienced programmers to combine REXX instructions and host commands and services into programs called REXX execs. It is a versatile programming language, with some aspects, such as common programming structure, readability, and free format, that make it a useful language for general users. The REXX language can be intermixed with commands to various host environments, provides powerful functions, and has extensive mathematical capabilities.

The TSO/E implementation of the REXX language enables REXX execs to run in any z/OS address space. You can write a REXX exec that includes TSO/E services and run it in a TSO/E address space, or you can write an application in REXX to run outside of a TSO/E address space.

> **Note:** There is also a set of z/OS UNIX extensions to the TSO/E REXX language that enable REXX programs to access z/OS UNIX callable services. The z/OS UNIX extensions, called *syscall commands*, have names that correspond to the names of the callable services that they start, for example, `access`, `chmod`, and `chown`. For more information about the z/OS UNIX extensions, see *z/OS Using REXX and z/OS UNIX System Services*, SA23-2283.

CLIST

An easy way to run a series of Session Manager or TSO/E commands is by using command procedures (CLISTs). A CLIST is an executable sequence of TSO/E and Session Manager commands, subcommands, or command procedure statements. You can use any TSO/E or Session Manager command in CLISTs. The same rules for entering Session Manager commands at your terminal apply when using a CLIST.

When started, CLIST issues the TSO/E commands in sequence. The CLIST language includes the programming tools that you need to write extensive, structured applications. CLISTs can perform several complex tasks, from displaying a series of full-screen panels to managing programs written in other languages.

You can include TSO/E commands and subcommands (and user-written commands and subcommands) in a CLIST at any point where the specific functions (for example, allocate, free, and so on) are required. For certain applications, a CLIST might consist entirely of commands and subcommands.

3.2 Interactive System Productivity Facility

The Interactive System Productivity Facility (ISPF) product assists in program development and is designed to take advantage of the characteristics of IBM display terminals, to increase programmer productivity in an interactive environment. It is a set of panels that help you manage libraries of information of the z/OS system. The libraries are made up of units called *data sets* that can be stored and retrieved. You can have various kinds of information in data sets. Several examples are shown here:

- Source code
- Data, such as inventory records, personnel files, or a series of numbers to be processed
- Load modules

For more information, see *z/OS ISPF User's Guide*, GC19-3627.

3.2.1 Using ISPF

ISPF is a multifaceted development tool set for the z/OS operating system. TSO/E users use ISPF for application development productivity. ISPF forms the basis of many TSO/E applications and provides extensive programmer-oriented facilities as well.

ISPF can be used in many ways, as these examples illustrate:

- Users can edit, browse, and print data.
- Data processing administrators and system programmers can use ISPF to:
  - Monitor and control program libraries.
  - Communicate with z/OS through TSO commands, CLISTs, or REXX execs.
- Programmers can use ISPF to develop a batch, interactive, or any other type of program and its documentation.
- Terminal users can start a wide range of utilities, such as search, compare, compile, and so on.

3.2.2 ISPF structure

ISPF helps programmers develop interactive applications called *dialogs*. Dialogs are interactive because ISPF uses them to communicate with terminal users through a series of panels while the users perform application development tasks.

ISPF panels provide the following functions:

- Provide access to ISPF functions through menus.
- Request information from users through data entry panels.
- Provide information from users through scrollable data displays.

These are the main components of ISPF:

- Dialog Manager (DM)

  DM provides services to dialogs and users. These include display, variable services, input and output, user and application profiles, table management, system interface services, dialog testing and debugging aids, and other services.

- Program Development Facility (PDF)

  PDF provides services to assist the dialog or application developer. These include the edit and browse functions, data set and catalog utilities, TSO/E command interfaces, and data set search and compare functions.

- Software Configuration and Library Manager (SCLM)

  The SCLM facility provides library management capabilities, such as versioning, auditing, and promotion. It also provides configuration management capabilities to track how all of the pieces of an application fit together, including source code, objects, load modules, test cases, documentation, and other items.

- Client/server component

  The client/server component enables the users of ISPF applications to use a workstation running Windows or UNIX to display the panels of an ISPF application. It does this using the graphical user interface (GUI) of the workstation.

3.2.3 Data set types supported

A *data set* is a collection of logically related data; it can be a source program, a library of macros, or a file of data records used by a processing program. *Data records* are the basic unit of information used by a processing program. ISPF supports the following data set types for any ISPF options, such as Edit, Browse, and Delete:

- Sequential data set

  A sequential data set is a data set whose records are organized based on their successive physical positions. In a sequential data set, records are stored and retrieved in a sequential order.

- Partitioned data set and partitioned data set extended

  A partitioned data set (PDS) is a data set on disk that is divided into partitions called *members*. Each member can contain a program, part of a program, or data.

  A *partitioned data set* is like a collection of sequential data sets, where the individual members each have a unique name. A directory index is used to locate members in the partitioned data set. The directory consists of 256-byte records, each one containing directory entries. There is one directory entry for each member.

ISPF provides partial support for Virtual Storage Access Method (VSAM) data sets and tape data sets.

- You can create and delete VSAM data sets and obtain VSAM data set information.
- VSAM data sets are supported for Edit, Browse, and View if the ISPF has been customized to enable such support.
- You can define an interface to an external utility such as DFSMSrmm that the Data Set List utility can use to process data sets stored on tape or some other removable media.

ISPF supports z/OS UNIX directories and files in the ISPF Edit and Browse options, as well as in the ISPF services BROWSE, EDIT, and VIEW. It supports processing of directories and files in a z/OS UNIX directory structure.

ISPF does *not* support some of the following data set types:

- Record format variable block spanned (VBS) data sets
- Direct access (DA) data sets
- Generation data group (GDG) base data sets

3.2.4 ISPF components

ISPF consists of four major components, DM, PDF, SCLM, and client/server. These components are considered one element in all releases of z/OS.

ISPF Dialog Manager

DM provides services to dialogs and users. A *dialog* is the interaction between a person and a computer. It helps a person who is using an interactive display terminal to exchange information with a computer. The user starts an interactive application through an interface that the system provides. The dialog with the user begins with the computer displaying a panel and asking for user interaction. It ends when the task for which the interactions were initiated is completed.

ISPF provides facilities to the user to create the parts of a dialog, called dialog elements. Each dialog application is made up of a command procedure or program, together with dialog elements that allow an orderly interaction between the computer and the application user.

The elements that make up a dialog application are:

- Functions. A function is a command procedure or a program that performs processing requested by the user. It can start ISPF dialog services to display panels and messages, build and maintain tables, generate output data sets, and control operational modes. They can be written as:
  - REXX or CLIST command procedures
  - Programs
- Panel definitions. A panel definition is a programmed description of the panel. It defines both the content and format of a panel. Most panels prompt the user for input. The user's response can identify which path is to be taken through the dialog, as on a selection panel. The response can be interpreted as data, as on a data-entry panel.
- Message definitions. Message definitions specify the format and text of messages to users. A message can confirm that a user-requested action is in progress or completed, or it can report an error in the user's input.
- Table. Tables are two-dimensional arrays that contain data and are created by dialog processing. They can be created as a temporary data repository, or they can be retained across sessions. A retained table can also be shared among several applications. The type and amount of data stored in a table depends on the nature of the application. Not all dialogs use tables.

- File tailoring skeletons. Skeletons work like a fill-in-the-blank exercise. They take dialog variables and put them into a data set containing statements that control the output format. The output data set can be used to drive other processes. File skeletons are frequently used to produce job data sets for batch execution. Various dialogs can use this resource.
- Dialog variables. ISPF services use variables to communicate information among the various elements of a dialog application. ISPF provides a group of services for variable management. Variables can vary in length from 0 - 32,000 bytes.

Program Development Facility

Program Development Facility (PDF) provides View, Browse, Edit, and library access services that can be combined in a dialog with any of the ISPF services. The library access services run functions involving members of a programming library. These functions include adding, finding, and deleting members, and displaying member lists.

Software Configuration Library Manager

Software Configuration Library Manager (SCLM) is a software tool that helps you develop complex software applications. Throughout the development cycle, SCLM automatically controls, maintains, and tracks all of the software components of the application. You can lock the version being edited in a private library and then promote it. Use SCLM to create, control, maintain, and track software components for a project. For more information about SCLM, see *z/OS ISPF Software Configuration and Library Manager Guide and Reference*, SC19-3625.

ISPF client/server - the Workstation Agent component

The client/server component of ISPF takes the form of an application called the ISPF Workstation Agent (WSA). The WSA runs on your local workstation and maintains a connection between the workstation and the ISPF host. The WSA provides:

- The ability to display the ISPF panels using the display function of your workstation operating system (known as running in GUI mode).
- The ability to edit host data on your workstation and workstation data on the host (known as distributed editing).

The WSA installation file is supplied with ISPF and must be downloaded from the host to the local workstation and then installed and initialized before these functions are available. The WSA component enables you to run ISPF on a programmable workstation and display the panels using the display function of your workstation operating system. Manuals in the ISPF library refer to this as *running in GUI mode*. The ISPF WSA is supported on the following platforms:

- Microsoft Windows
- IBM AIX®
- HP-UX
- Solaris

Connecting to a workstation for data access has a direct effect on your installation's processor processing time. One reason for using the ISPF client/server function is to offload processor cycles from the host to a less expensive workstation. But even if that is not your goal, an added benefit is that your users can use the connection for distributed editing.

Therefore, they can use their favorite editor to work with your data, whether that means using a host editor on host and workstation files, or using a workstation editor on the same files. By making the connection to the workstation, a user can edit workstation files on ISPF, or host files on his workstation. The distributed edit function can be used in standard 3270 mode, or in ISPF GUI mode.

3.2.5 ISPF primary option menu

ISPF is started in a TSO/E environment through an ISPF, or PDF, or ISPSTART command. The ISPF Primary Option Menu contains the options that you can use to create your own applications online. If your installation has a customized ISPF Primary Option Menu, the menu might not contain all of options shown in Figure 3-5, or it might contain certain installation-specific options. Most ISPF panels have action bars at the top; many panels also have point-and-shoot text fields.

```
   Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
                          ISPF Primary Option Menu
 Option ===> _____________________________________________________________________

 0  Settings      Terminal and user parameters           User ID . : LUIZ
 1  View          Display source data or listings        Time. . . : 08:24
 2  Edit          Create or change source data           Terminal. : 3278
 3  Utilities     Perform utility functions              Screen. . : 1
 4  Foreground    Interactive language processing        Language. : ENGLISH
 5  Batch         Submit job for language processing     Appl ID . : PDF
 6  Command       Enter TSO or Workstation commands      TSO logon : IKJACCT
 7  Dialog Test   Perform dialog testing                 TSO prefix: LUIZ
 9  IBM Products  IBM program development products       System ID : SC74
 10 SCLM          SW Configuration Library Manager       MVS acct. : ACCNT#
 11 Workplace     ISPF Object/Action Workplace           Release . : ISPF 7.3
 12 z/OS System   z/OS system programmer applications
 13 z/OS User     z/OS user applications

      Enter X to Terminate using log/list defaults
```

Figure 3-5 ISPF primary option menu

The following list describes the panel options:

0 **Settings** displays and changes selected ISPF parameters, such as terminal characteristics and function keys.

1 **View** displays data using the View or Browse function. You can use View or Browse to look at (but not change) large data sets, such as compiler listings. You can scroll the data up, down, left, or right. If you are using Browse, a **FIND** command, entered on the command line, enables you to search the data for a character string. If you are using View, you can use all the commands and macros available to you in the Edit function.

2 **Edit** enables you to create or change source data, such as program code and documentation, using the ISPF full-screen editor. You can scroll the data up, down, left, or right. You can change the data by using Edit line commands, which are entered directly on a line number, and primary commands, which are entered on the command line.

3 **Utilities** perform library and data set maintenance tasks, such as moving or copying library or data set members, displaying or printing data set names and volume table of contents (VTOC) information, comparing data sets, and searching for strings of data.

4 **Foreground** calls IBM language processing programs in the foreground.

5 **Batch** calls IBM language processing programs as batch jobs. ISPF generates JCL based on information that you enter and submits the job for processing.

6 **Command** calls TSO commands, CLISTs, or REXX execs under ISPF.

7 **Dialog Test** tests individual ISPF dialog components, such as panels, messages, and dialog functions (programs, commands, menus).

9 **IBM Products** enable you to select other installed IBM program development products on your system.

The following products are supported:

- Tivoli Information Management
- COBOL Structuring Facility foreground dialog (COBOL/SF)
- Screen Definition Facility II (SDF II) licensed program
- Screen Definition Facility II-P (SDF II-P) licensed program

10 **SCLM** controls, maintains, and tracks all of the software components of an application.

11 **Workplace** gives you access to the ISPF Workplace, which combines many of the ISPF functions onto one object-action panel.

12 **z/OS System** gives you access to the z/OS System Programmer Primary Option Menu. It contains options for z/OS elements that are used by system programmers and administrators as tailored by the installation:

- Graphical Data Display Manager (GDDM) Print Queue Manager
- Hardware Configuration Definition (HCD) I/O configuration
- DCE configuration
- APPC Administration
- z/OS Workload Manager (WLM)
- IBM First Failure Support Technology™ (FFST™)
- Infoprint Server
- z/OS Resource Management Facility (RMF)
- System Modification Program/Extended (SMP/E)
- Transmission Control Protocol/Internet Protocol (TCP/IP) Network Print Facility (NPF)

13 **z/OS User** gives you access to the z/OS Applications panel. It contains options for z/OS elements that are used by most ISPF users:

- IBM BookManager® Build
- BookManager Read
- BookManager Index Creation
- DFSMS Data Facility Product (DFSMSdfp)/ISMF
- Data Facility Sort (DFSORT)
- Bulk Data Transfer (BDT) File-to-File
- Interactive Problem Control System (IPCS)
- z/OS UNIX Browse
- z/OS UNIX Edit
- z/OS UNIX Shell
- Security Server
- TSO/E Information Center Facility
- SDSF
- Data Facility Storage Management Subsystem (DFSMS) Removable Media Manager (DFSMSrmm)/Interactive Storage Management Facility (ISMF)

X EXIT leaves ISPF using the log and list defaults. You can change these defaults from the Log/List pull-down on the ISPF Settings panel action bar.

3.3 Job control language

For your program to run on the computer and perform the work that you designed it to do, your program must be processed by your operating system.

Your operating system consists of a z/OS Base Control Program (BCP) with a job entry subsystem (JES2 or JES3) and DFSMSdfp installed with it.

For the operating system to process a program, programmers must perform certain job control tasks. These tasks are performed through the job control statements:

- JCL statements
- JES2 control statements
- JES3 control statements

The JES2 and JES3 statements are called Job Entry Control Language (JECL) statements.

3.3.1 JCL introduction

To get your z/OS system to accomplish work for you, you must describe to the system the work you want done and the resources your work needs. You use JCL to provide this information to z/OS.

One way of thinking about JCL is to compare it to a menu in a restaurant. If you are a customer at a restaurant, you and the other customers do not walk into the kitchen and start cooking your own dinners; that defeats the purpose of going to a restaurant. Rather, you select items from a menu describing all that the restaurant has to offer. These items make up an order, specifying which entrées you want, which salad dressing you prefer, and any other special requests you have. You then ask the waiter to take your order to the kitchen.

In the kitchen, a team of chefs divides up the work and the appropriate ingredients to prepare each dish as quickly and efficiently as possible. While the meals are being prepared, you and your friends can ignore what is going on in the kitchen, engaging instead in dinner conversation and catching up on the latest news. When the waiter brings out your meal, you concentrate on your enjoyment of the meal.

Now imagine yourself back at the office using your z/OS system, and think of JCL as the menu. In the same way that you and the other diners select items from the menu and place orders for the waiter to take to the team of chefs, you and other z/OS users use JCL to define work requests (called jobs), and use a JES to submit those jobs to z/OS.

Using the information that you and the other users provide with JCL statements, z/OS allocates the resources needed to complete all of your jobs just as the kitchen chefs divided up the work to prepare the orders of all the customers.

In addition, just as the chefs worked in the kitchen while you and the other diners devoted your attention to what was going on at your tables, z/OS completes the submitted jobs in the background of the system, enabling you and the other users to continue working on other activities in the foreground. Also, just as the waiter conveys the results of the chefs' work to you, JES presents the output of the jobs to you.

For a complete description of this process, see *z/OS JCL User's Guide*, SA23-1386.

Job submission process

Figure 3-6 shows an overview of the job submission process. The user performs the activities on the left side of the figure, and the system performs those on the right side. In this example, z/OS and JES make up *the system*.

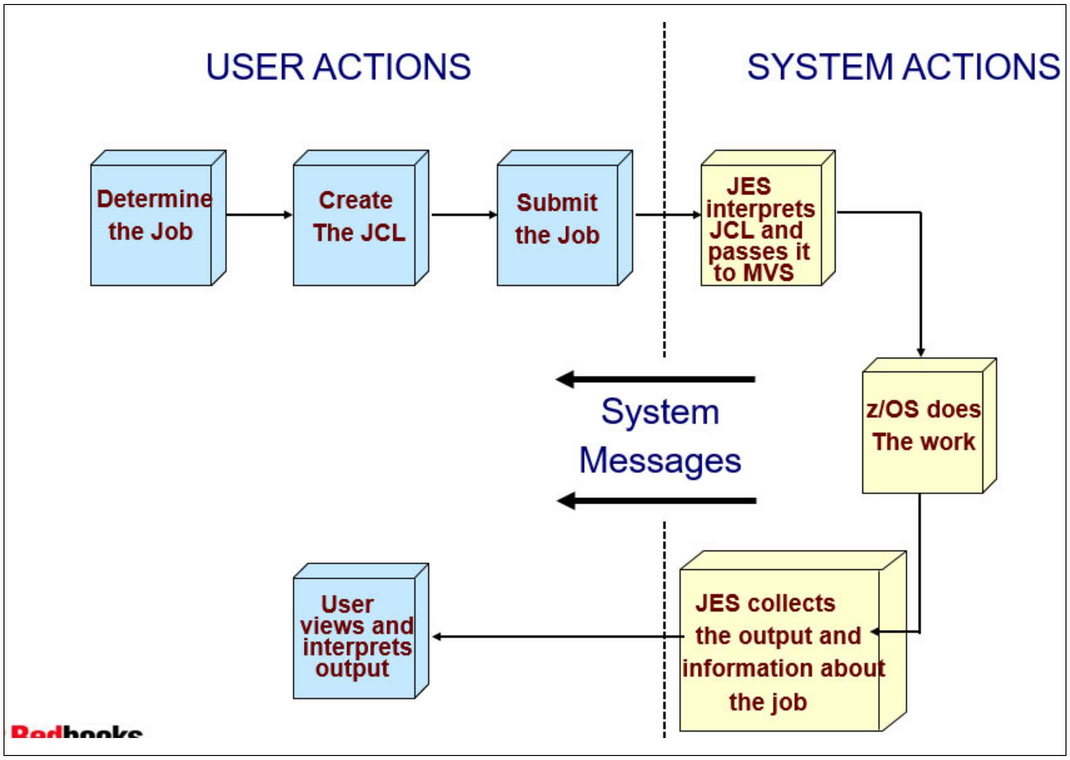

Figure 3-6 Job Submission Process

For every job that you submit, you need to tell z/OS where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output.

Job control statements

You use JCL to convey this information to z/OS through a set of statements known as *job control statements*. JCL's set of job control statements is quite large, enabling you to provide a great deal of information to z/OS.

Most jobs, however, can be run using a small subset of these control statements. After you become familiar with the characteristics of the jobs you typically run, you might find that you need to know the details of only some of the control statements.

In each job, the control statements are grouped into job steps. A job step consists of all the control statements needed to run one process, for example a sort, a copy, or an application program. If a job needs to run more than one process, the job contains another job step for each of those programs. A job can have 1 - 255 steps.

Required control statements

Every job must contain, at minimum, the following two types of control statements:

- A JOB statement

  This statement marks the beginning of a job and assigns a name to the job. The JOB statement is also used to provide certain administrative information, including security, accounting, and identification information. Every job has one and only one JOB statement.

- An EXEC (run) statement

  This statement marks the beginning of a job step, to assign a name to the step and identify the program or procedure to be run in the step. You can add various parameters to the EXEC statement to customize the way the program runs. Every job has at least one EXEC statement.

In addition to the JOB and EXEC statements, most jobs also contain one or more data definition (DD) statements. These statements identify and describe the input and output data to be used in the step. The DD statement can be used to request a previously created data set, to define a new data set, to define a temporary data set, or to define and specify the characteristics of the output.

3.3.2 JCL streams and jobs

For the operating system to process a program, system programmers or application programmers must perform certain job control tasks. These tasks are performed through the job control statements:

- JCL statements, as mentioned previously
- JES2 control statements *or* JES3 control statements

3.3.3 Job Entry Subsystems

z/OS uses a job entry subsystem or JES to receive jobs into the operating system, to schedule them for processing by z/OS, and to control their output processing.

JES is the component of the operating system that provides supplementary job management, data management, and task management functions such as scheduling, control of job flow, and the reading and writing of input and output streams on auxiliary storage devices, concurrently with job execution.

z/OS manages work as tasks and subtasks. Both transactions and batch jobs are associated with an internal task queue that is managed on a priority basis. JES is a component of z/OS that works on the front end of program execution to prepare work to be run. JES is also active on the back end of program execution to help clean up after work is performed. This activity includes managing the printing of output generated by active programs.

More specifically, JES manages the input and output job queues and data.

For example, JES handles the following aspects of batch processing for z/OS:

- Receiving jobs into the operating system
- Scheduling them for processing by z/OS
- Controlling their output processing

z/OS has two versions of job entry systems: JES2 and JES3. Of these, JES2 is the most common by far.

JES2 and JES3 have many functions and features, but their most basic functions are as follows:

- Accept jobs submitted in various ways:
  - From ISPF through the SUBMIT command
  - Over a network
  - From a running program, which can submit other jobs through the JES internal reader
  - From a card reader (very rare!)
- Queue jobs waiting to be run. Multiple queues can be defined for various purposes.
- Queue jobs for an initiator, which is a system program that requests the next job in the appropriate queue.
- Accept printed output from a job while it is running and queue the output.
- Optionally, send output to a printer, or save it on spool for PSF, InfoPrint, or another output manager to retrieve.

The basic elements of batch processing are shown in Figure 3-7.

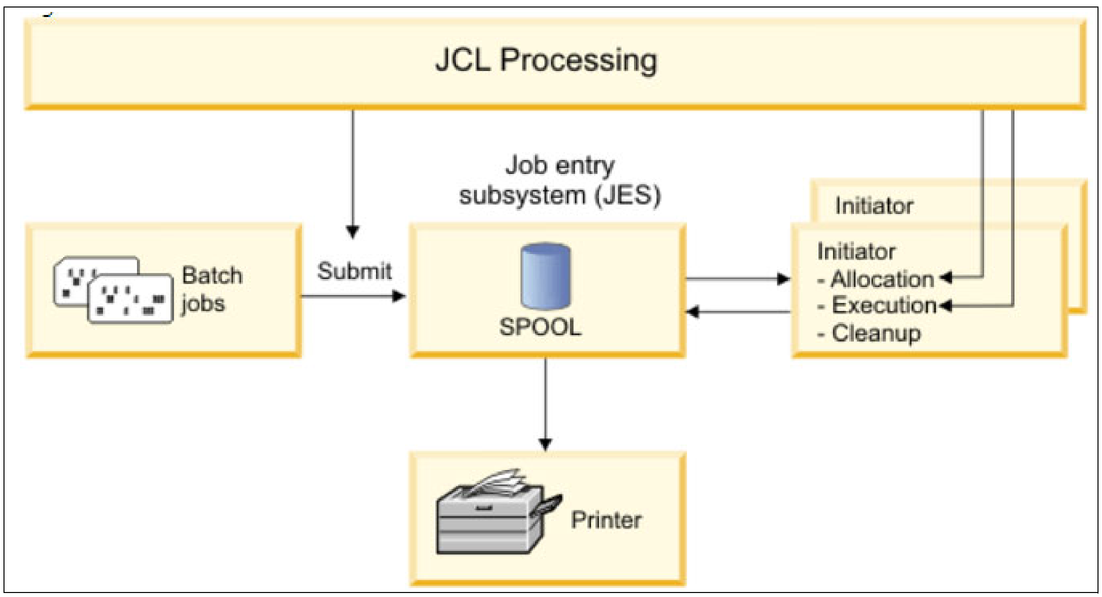

Figure 3-7 Batch Job Flow

JES uses one or more disk data sets for spooling, which is the process of reading and writing input and output streams on auxiliary storage devices, concurrently with job execution, in a format convenient for later processing or output operations. *Spool* is an acronym that stands for simultaneous peripheral operations online.

JES combines multiple spool data sets (if present) into a single conceptual data set. The internal format is not in a standard access-method format, and is not written or read directly by applications. Input jobs and printed output from many jobs are stored in the single (conceptual) spool data set. In a small z/OS system, the spool data sets might be a few hundred cylinders of disk space; in a large installation, they might be many complete volumes of disk space.

There are two versions of JES: JES2 and JES3.

3.3.4 Job Entry Subsystem 2

JES2 is a component of z/OS that provides the necessary functions to get jobs into, and output out of, the z/OS system. It is designed to provide efficient spooling, scheduling, and management facilities for the z/OS operating system.

During the life of a job, JES2 and z/OS control different phases of the overall processing.

The job queues contain jobs that are waiting to run, currently running, waiting for their output to be produced, having their output produced, and waiting to be purged from the system.

Generally speaking, a job goes through the following phases, as shown in Figure 3-8:

1. Input
2. Conversion
3. Processing
4. Output
5. Print/punch (hardcopy)
6. Purge

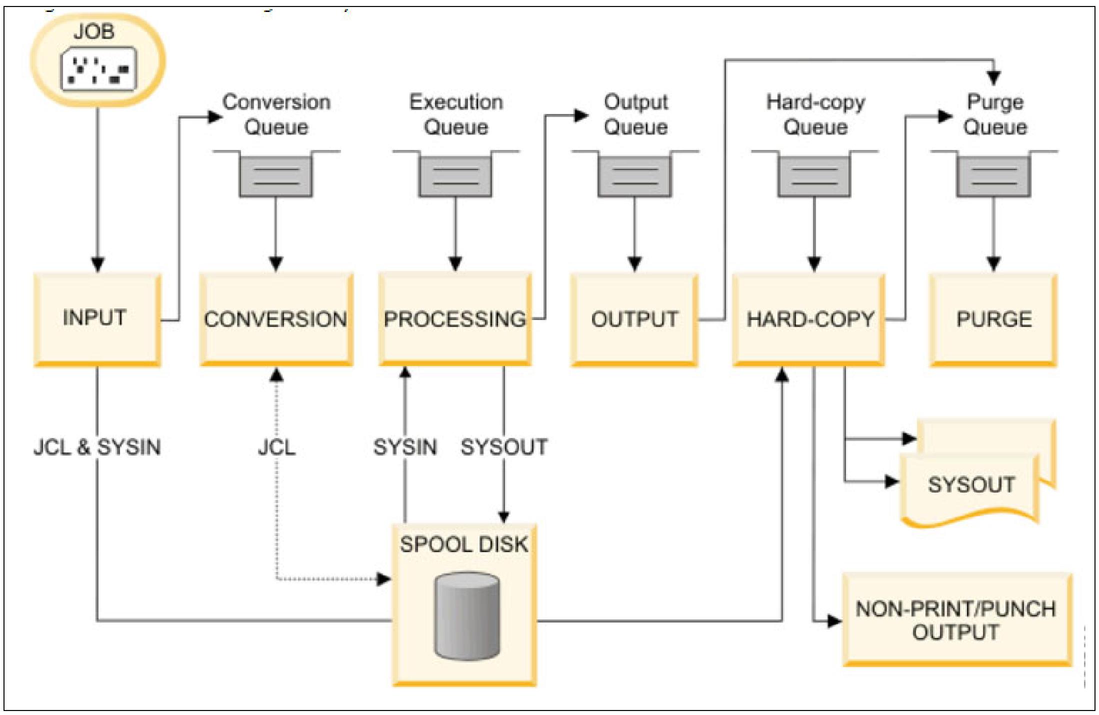

Figure 3-8 Job phases through JES2

Input phase

JES2 accepts jobs, in the form of an input stream, from input devices, from other programs through internal readers, and from other nodes in a job entry network.

The internal reader is a program that other programs can use to submit jobs, control statements, and commands to JES2. Any job running in z/OS can use an internal reader to pass an input stream to JES2. JES2 can receive multiple jobs simultaneously through multiple internal readers. The system programmer defines internal readers to be used to process all batch jobs other than started tasks (STCs) and TSO requests.

JES2 reads the input stream and assigns a job identifier to each JOB JCL statement. JES2 places the job's JCL, optional JES2 control statements, and SYSIN data onto DASD data sets called spool data sets. JES2 then selects jobs from the spool data sets for processing and subsequent running.

Conversion phase

JES2 uses a converter program to analyze a job's JCL statements. The converter takes the job's JCL and merges it with JCL from a procedure library. The procedure library can be defined in the JCLLIB JCL statement, or system/user procedure libraries can be defined in the PROCxx DD statement of the JES2 startup procedure.

Then, JES2 converts the composite JCL into converter/interpreter text that both JES2 and the initiator can recognize. Next, JES2 stores the converter/interpreter text on the spool data set. If JES2 detects any JCL errors, it issues messages, and the job is queued for output processing rather than execution. If there are no errors, JES2 queues the job for execution.

Processing phase

In the processing phase, JES2 responds to requests for jobs from the initiators. JES2 selects jobs that are waiting to run from a job queue and sends them to initiators.

An initiator is a system program belonging to z/OS, but controlled by JES or by the workload management (WLM) component of z/OS, which starts a job allocating the required resources to allow it to compete with other jobs that are already running.

JES2 initiators are initiators that are started by the operator or by JES2 automatically when the system initializes. They are defined to JES2 through JES2 initialization statements. To obtain an efficient use of available system resources, the installation associates each initiator with one or more job classes. Initiators select jobs whose classes match the initiator-assigned class, obeying the priority of the queued jobs.

WLM initiators are started by the system automatically based on performance goals, relative importance of the batch workload, and the capacity of the system to do more work. The initiators select jobs based on their service class and the order in which they were made available for execution. Jobs are routed to WLM initiators through a JOBCLASS JES2 initialization statement.

Output phase

JES2 controls all SYSOUT processing. SYSOUT is system-produced output; that is, all output produced by, or for, a job. This output includes system messages that must be printed, and data sets requested by the user that must be printed or punched. After a job finishes, JES2 analyzes the characteristics of the job's output in terms of its output class and device setup requirements. Then, JES2 groups data sets with similar characteristics. JES2 queues the output for print or punch processing.

Print/punch (hardcopy) phase

JES2 selects output for processing from the output queues by output class, route code, priority, and other criteria. The output queue can have output that is to be processed locally or at a remote location. After processing all the output for a particular job, JES2 puts the job on the purge queue.

Purge phase

When all processing for a job completes, JES2 releases the spool space assigned to the job, making the space available for allocation to subsequent jobs. JES2 then issues a message to the operator indicating that the job has been purged from the system.

3.3.5 Job Entry Subsystem 3

A major goal of operating systems is to process jobs while making the best use of system resources. Thus, one way of viewing operating systems is as resource managers. Before job processing, operating systems reserve input and output resources for jobs. During job processing, operating systems manage resources such as processors and storage. After job processing, operating systems free all resources used by the completed jobs, making the resources available to other jobs. This process is called *resource management*.

There is more to the processing of jobs than the managing of resources needed by the jobs. At any instant, a number of jobs can be in various stages of preparation, processing, and post-processing activity. To use resources efficiently, operating systems divide jobs into parts. They distribute the parts of jobs to queues to wait for needed resources. Keeping track of where things are and routing work from queue to queue is called *workflow management*, and is a major function of any operating system.

With the z/OS JES3 system, resource management and workflow management are shared between z/OS and its Job Entry Subsystem 3 (JES3) component. Generally speaking, JES3 does resource management and workflow management before and after job execution, while z/OS does resource and workflow management during job execution.

JES3 considers job priorities, device and processor alternatives, and installation-specified preferences in preparing jobs for processing job output. Features of the JES3 design include:

- Single-system image
- Workload balancing
- Availability
- Control flexibility

Physical planning flexibility In contrast, JES3 exercises centralized control over its processing functions through a single global JES3 processor. This global processor provides all job selection, scheduling, and device allocation functions for all the other JES3 systems. The centralized control that JES3 exercises provides increased job scheduling control, deadline scheduling capabilities, and increased control by providing its own device allocation.

JES3 runs on either 1 processor or up to 32 processors in a sysplex. A *sysplex* is a set of z/OS systems communicating and cooperating with each other through certain multisystem hardware components and software services to process customer workloads.

In a sysplex, your installation must designate one processor as the focal point for the entry and distribution of jobs, and for the control of resources needed by the jobs. That processor, called the *global processor*, distributes work to the processors, called *local processors*.

It is from the global processor that JES3 manages jobs and resources for the entire complex, matching jobs with available resources. JES3 manages processors, I/O devices, volumes, and data. To avoid delays that result when these resources are not available, JES3 ensures that they are available before selecting the job for processing.

JES3 keeps track of I/O resources, and manages workflow in conjunction with the workload management component of z/OS by scheduling jobs for processing on the processors where the jobs can run most efficiently. At the same time, JES3 maintains data integrity. JES3 does not schedule two jobs to run simultaneously anywhere in the complex if they are going to update the same data.

If you want to share input/output (I/O) devices among processors, JES3 manages the sharing. Operators do not have to manually switch devices to keep up with changing processor needs for the devices.

The JES3 architecture of a global processor, centralized resource and workflow management, and centralized operator control is meant to convey a single-system image, rather than one of separate and independently operated computers.

The work environment for JES3 consists of processors and I/O devices. JES3 covers a range of data processing needs, partly because it can accommodate various combinations of processors and devices. JES3 is a manager of its environment.

Single-Processor JES3 Environment

Figure 3-9 shows JES3 in a single-processor environment, also known as a single-system sysplex. In addition to the processor, two categories of I/O devices are shown: JES3 devices (those used by JES3); and JES3-managed and z/OS-managed devices (those used by jobs). The spool device is a direct access storage device (DASD) that is treated in a special way by JES3, so it is shown and explained separately.

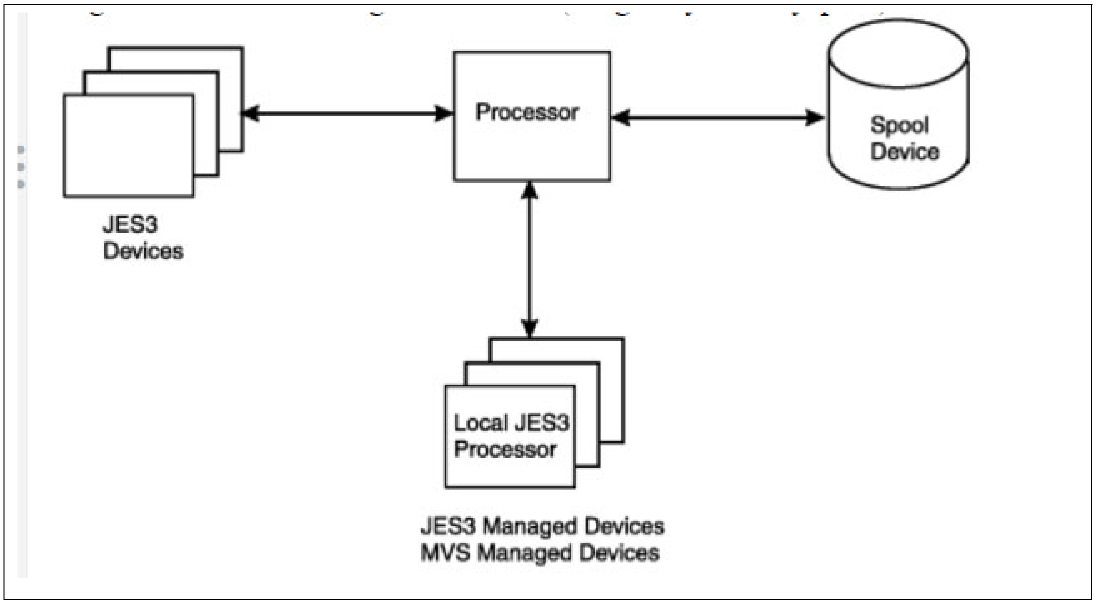

Figure 3-9 JES3 in a Single Processor Environment

Multiprocessor JES3 environment

Multiprocessing is a way of doing work with two or more connected processors. In a multiprocessing environment, also known as a multisystem sysplex, JES3 allows up to 32 processors, also known as mains, to be configured into the complex. JES3 uses one processor (called the global) to do work and also to distribute work to up to 31 other processors (called locals). Running JES3 in this environment provides the following advantages:

- Eliminating much of the resources required by scheduling work for and operating separate processors
- Sharing devices by processors, which means that the devices can be used more efficiently
- Moving work to other processors, should one processor become overworked, need maintenance, or need to be removed from the complex for any reason

Figure 3-10 shows a multiprocessor environment managed by JES3:

- Global processor: The processor that controls job scheduling and device allocation for a complex of processors.
- Local processor: In a complex of processors under control of JES3, a processor connected to the global main, for which JES3 performs centralized job input, job scheduling and job output services by the global main.
- Global Main: The global main controls job scheduling and device allocation for a complex of JES3 processors. Each local main in the complex exists under control of the JES3 global main and is connected to the global main. The JES3 on the global main can perform centralized job input, job scheduling, and job output services. Only the global main performs scheduling functions, although scheduled work runs on the local mains.
- Local Main: In a complex of processors under control of JES3, a processor connected to the global main, for which JES3 performs centralized job input, job scheduling, and job output services by the global main.

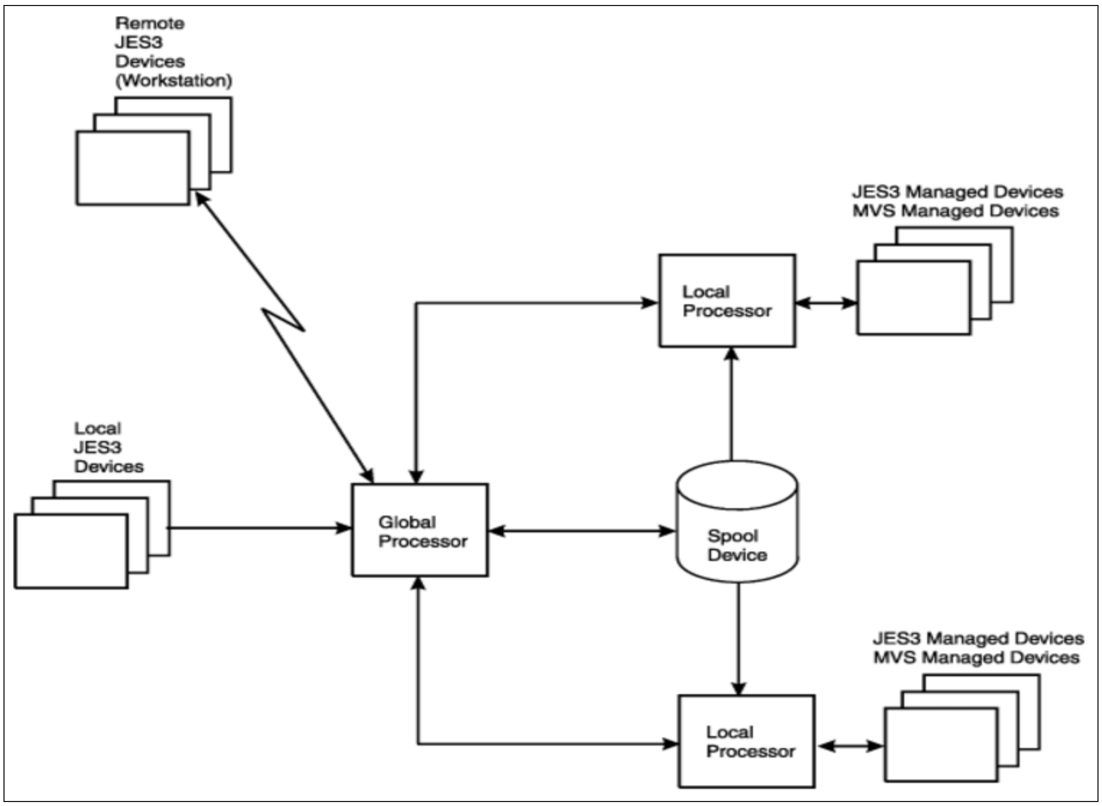

Figure 3-10 JES3 Multiprocessor environment

JES3 job flow

This section describes the JES3 job flow.

Input service

Whatever the source of the input, JES3 is signaled that an input stream is to be read, as shown in Figure 3-11 on page 53. This begins a chain of events that includes:

- Creating and scheduling a card reader DSP: a JES3 job.
- Reading the input stream by a DSP.

- Building the JES3 control blocks describing the z/OS job to JES3. JCT entries for each job in the input stream are added to the JES3 JCT data set.
- Running DSPs represented by scheduler elements in the JCT entries for each job.

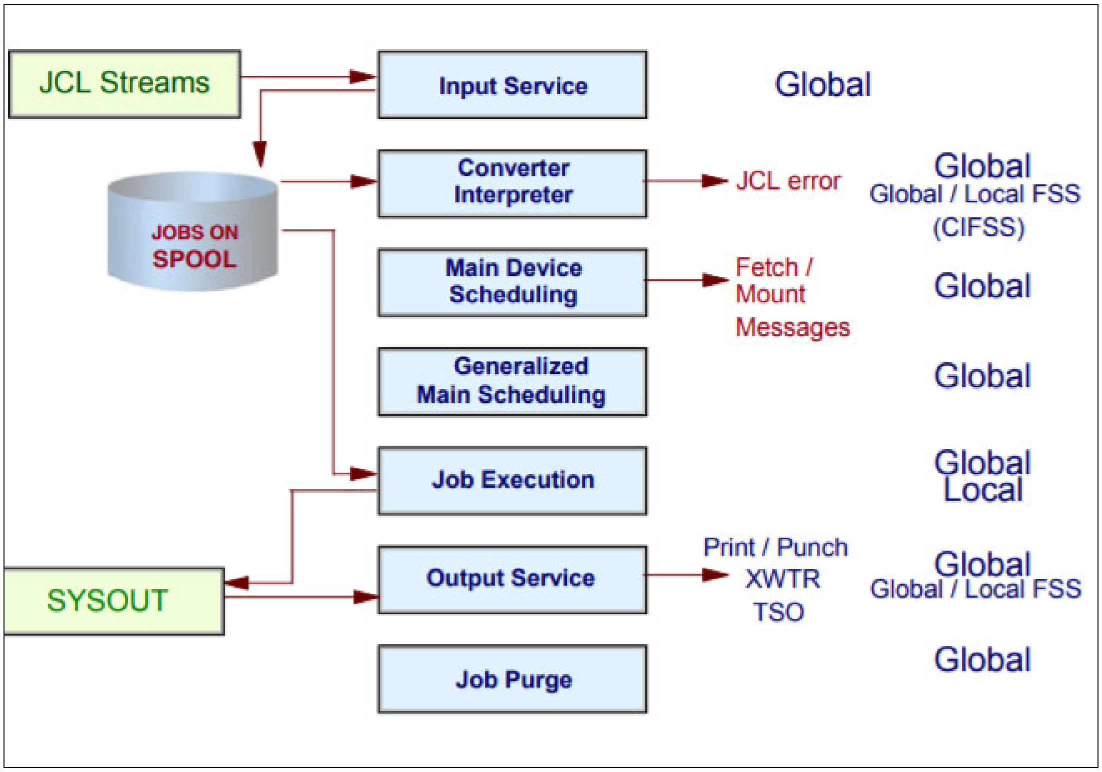

Figure 3-11 JES3 Job Flow

The DSPs that provide the JES3 input service control the processing of a typical z/OS job at the beginning. Input service routines create scheduler elements that represent a job's flow through the various JES3 processing phases. Input service, active on the global processor, accepts and queues all jobs entering the JES3 complex. The global processor accepts z/OS jobs into the system from:

- TSO **SUBMIT** command
- Local card reader (CR DSP)
- Local tape reader (TR DSP)
- Disk reader (DR DSP)
- Remote work station (RJP/SNARJP DSPs)
- Another node in a job entry network (NJE DSPs)
- The internal reader (INTRDR DSP)

After a z/OS job is registered into the JES3, the JES3 job segment scheduler DSP (JSS) becomes responsible for the flow of the job through the JES3 processing phases.

Converter interpreter processing

The converter/interpreter (C/I) is the first scheduler element for every standard job. After a job passes through this phase of processing, JES3 knows what resources the job will require during execution. C/I routines provide input to the main device scheduling (MDS) routines by determining jobs' devices, volumes, and data sets requirements. The C/I routines, that run after the z/OS converter/interpreter invocation, extract from the jobs' scheduler work area (SWA) the jobs' resource requirements and create control blocks for MDS. Jobs with JCL errors are flushed. Main device scheduling provides for the effective use of system resources.

Main device scheduling (MDS)

JES3 MDS, commonly referred to as “setup”, ensures the operative use of non-sharable mountable volumes, eliminates operator intervention during job execution, and performs JES3 data set serialization. It oversees specific types of pre-execution job setup and generally prepares all necessary resources required to run the job. The resource tables and allocation algorithms to satisfy a job’s requirements through the MDS allocation of volumes and devices are established at JES3 initialization.

Generalized main scheduling (GMS)

JES3 generalized main scheduling (GMS) is the group of routines that govern where and when z/OS execution of a job occurs. Job scheduling controls the order and execution of jobs running within the JES3 complex.

Job execution

Job execution is under the control of JES3 main service, which selects jobs to be processed by z/OS initiators. Main service selects a job for execution using the job selection algorithms established at JES3 initialization. MAINPROC, SELECT, CLASS, and GROUP initialization statements control the key variables in the job scheduling and job execution process.

Output service

Output service routines operate in various phases to process sysout data sets destined for local or remote print or punch devices, TSO users, external writers, and writer functional subsystems.

Job purge

Purge processing represents the last scheduler element for all JES3 job. That is, the last processing phases for the jobs. It releases the JES3 resources allocated for the job and cuts the System Management Facility (SMF) to records.

JES control statements in JCL

Job Entry Control Language or JECL is the set of command language control statements that provide information for JES2 or JES3. JECL statements can specify which z/OS to run the job, when to run the job, and where to send the resulting output.

JECL is distinct from job control language (JCL), which instructs the operating system how to run the job. JES2 and JES3 are different. JES2 can process JES3 JECL control statements. JES2 provides a new set of commands to control how JES2 treats JES3 JECL. You can ignore all JES3 JECL statements, or selectively decide which commands are processed, ignored, or failed.

For JES2 JECL statements start with `/*`, for JES3 they start with `//*`, except for remote **`/*SIGNON`** and **`/*SIGNOFF`** commands.

3.4 System Display and Search Facility

System Display and Search Facility (SDSF) is a licensed utility that allows you to monitor, control, and view the output of jobs in the system.

After submitting a job, it is common to use System Display and Search Facility (SDSF) to review the output for successful completion or to review and correct JCL errors. SDSF allows you to display printed output held in the JES spool area. Much of the printed output sent to JES by batch jobs (and other jobs) is never actually printed. Rather, it is inspected using SDSF and deleted or used as needed.

With SDSF you can perform the following tasks:

- Viewing the system log and searching for any literal string
- Entering system commands
- Control job processing (hold, release, cancel, and purge jobs)
- Monitor jobs while they are being processed
- Display job output before deciding to print it
- Manage the system's workflow
- Control the order in which jobs are processed
- Determine the number of output jobs and the total number of records to be printed
- Control the order in which output is printed
- Control printers and initiators
- Dynamically change job data set output descriptors
- Issue JES and z/OS commands that affect their jobs
- Print selected lines of the JES output data set
- Edit JCL direct from spool

SDSF function is available in IBM z/OS Management Facility (z/OSMF), which enables you to manage aspects of z/OS through a web browser interface. You can see system activity at a glance with graphic views of CPU use; manage and browse jobs, output, and checks for IBM Health Checker for z/OS; and issue system commands and export SDSF tables for use by other programs, such as spreadsheet applications.

3.4.1 SDSF: Panels hierarchy

SDSF consists of panels that provide immediate information about jobs, printers, queues, and resources in a JES2 system. The SDSF panel hierarchy is illustrated in Figure 3-12. From these panels, authorized users can enter SDSF commands to control the processing of jobs and the operation of system resources. Authorized users also can issue z/OS and JES2 system commands from the SDSF panels.

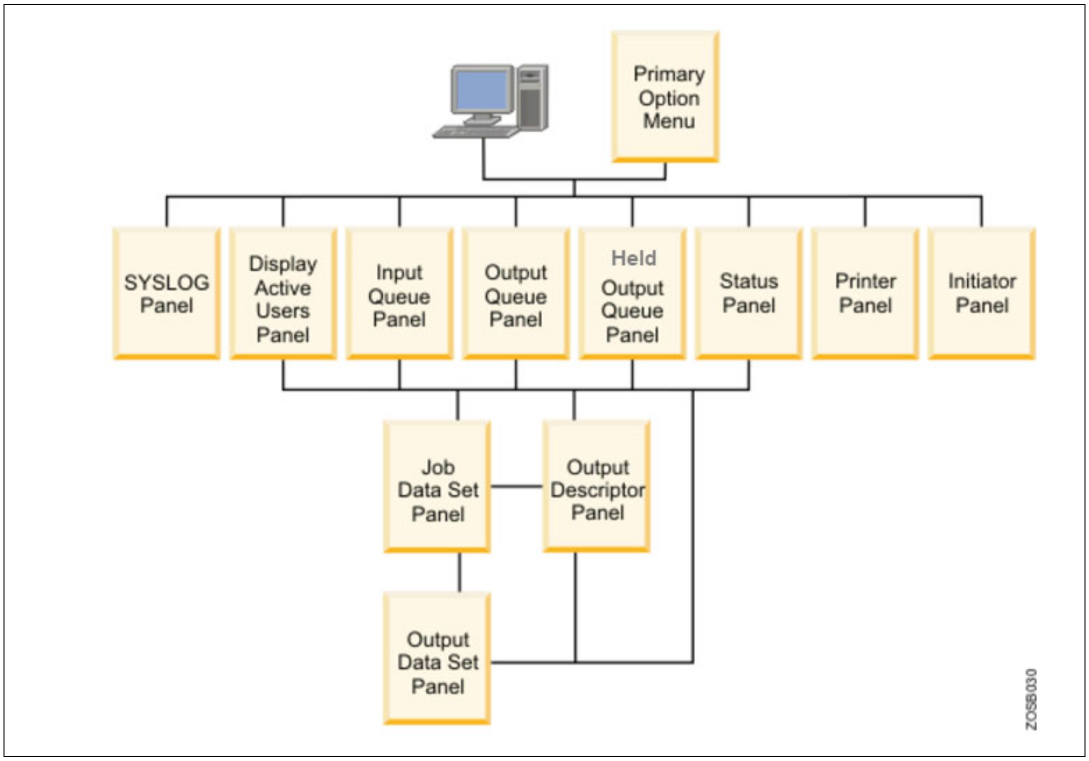

Figure 3-12 SDSF: JES2 Panel hierarchy

SDSF provides an easy way to manage JES2 jobs, which can help you work more efficiently. It gives immediate, current, sysplex-wide information about jobs waiting to be processed or in execution, such as the following items:

- The status, class, priority, date, and time of a specific job
- All jobs on a specific queue, such as the input or held output queue
- Detail for a job no matter where it is in the sysplex
- Reasons why a job might be delayed
- Output from a job as it is created

By using the SDSF panels, SDSF commands, and action characters, and by typing over panel fields, you can hold or release jobs, cancel jobs, filter the jobs displayed to show just the jobs that interest you, or change a job's priority, class, or destination.

3.4.2 JES2 SDSF Primary Option Menu

SDSF can be started from ISPF menus, but the setting of the options is often customized by each site differently. You must review your site's ISPF menus to find the SDSF option. Alternatively, issuing the `TSO SDSF` command from the command line starts SDSF. After choosing this option, the panel you receive will be similar to the one in Figure 3-13 on page 57.

```
  Display  Filter  View  Print  Options  Search  Help
 -------------------------------------------------------------------------------
 SDSF MENU V2R3M0    PLEX75    SC74                       LINE 1-27 (49)
 COMMAND INPUT ===> _                                            SCROLL ===> PAGE
 NP    NAME      Description                Group     Status
       DA        Active users               Jobs
       I         Input Queue                Jobs
       O         Output Queue               Output
       H         Held output Queue          Output
       ST        Status of jobs             Jobs
       JG        Job groups                 JES
       SYM       System symbols             System
       LOG       System log                 Log
       SR        System requests            Log
       MAS       Members in the MAS         JES
       JC        Job classes                JES
       SE        Scheduling environments    WLM
       RES       WLM resources              WLM
       ENC       Enclaves                   WLM
       PS        Processes                  OMVS
       SYS       System information         System
       ENQ       Enqueues                   System

  Licensed Materials - Property of IBM
  5650-ZOS
  Copyright IBM Corp. 1981, 2017.
  Copyright Rocket Software, Inc. 2015, 2017.
  US Government Users Restricted Rights -
  Use, duplication or disclosure restricted
  by GSA ADP Schedule Contract with IBM Corp.

       SP        Spool volumes              JES
```

Figure 3-13 JES2 SDSF Primary Option Menu

However, it might not have all the same options shown in the figure; the options can vary according to the security level of the user. The authority to perform functions in these options also varies according to the security level of the user. It is possible to control most system functions by using the SDSF facility. The scope of the functions includes reviewing job output, controlling the processing of jobs (both their input and output), printer control, operator functions, and system administration.

3.4.3 SDSF: JES3 panel hierarchy

SDSF consists of panels that provide immediate information about jobs, printers, queues, and resources in a JES3 system. The SDSF panel hierarchy is illustrated in Figure 3-14. From these panels, authorized users can enter SDSF commands to control the processing of jobs and the operation of system resources. Authorized users also can issue z/OS and JES3 system commands from the SDSF panels.

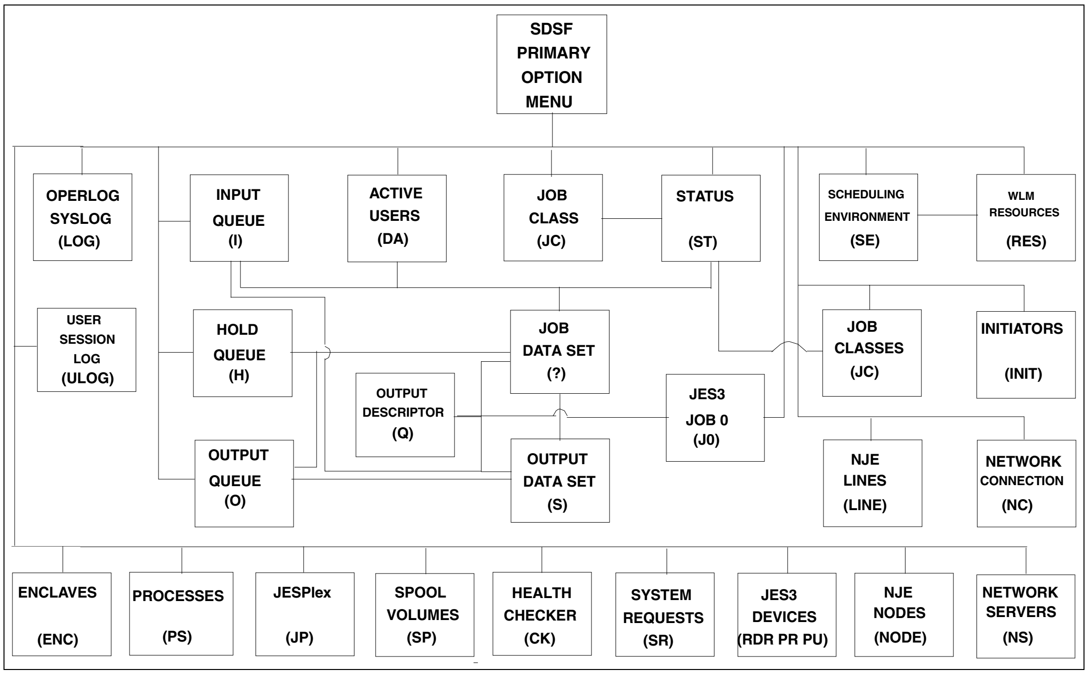

Figure 3-14 SDSF: JES3 panel hierarchy

SDSF provides an easy way to manage JES3 jobs, which can help you work more efficiently. It gives immediate, current, sysplex-wide information about jobs waiting to be processed or in execution, such as the following items:

- The status, class, priority, date and time of a specific job
- All jobs on a specific queue, such as the input or held output queue
- Detail for a job no matter where it is in the sysplex
- Reasons a job might be delayed
- Output from a job as it is created

By using the SDSF panels, SDSF commands and action characters, and by typing over panel fields, you can hold or release jobs, cancel jobs, filter the jobs displayed to show just the jobs that interest you, or change a job's priority, class, or destination.

3.4.4 JES3 SDSF Primary Option Menu

SDSF can be started from ISPF menus, but the setting of the options is often customized by each site differently. You must review your site's ISPF menus to find the SDSF option. Alternatively, issuing the **TSO SDSF** command from the command line starts SDSF. After choosing this option, the panel you receive will be similar to the one in Figure 3-15 on page 59.

```
  Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
HQX7780 -----------------  SDSF PRIMARY OPTION MENU  -------------------------
COMMAND INPUT ===>                                              SCROLL ===> HALF

DA     Active users                         INIT  Initiators
I      Input queue                          PR    Printers
O      Output queue                         PUN   Punches
H      Held output queue                    RDR   Readers
ST     Status of jobs                       LINE  Lines
J0     Job zero                             NODE  Nodes
                                            SP    Spool volumes
LOG    System log                           NS    Network servers
SR     System requests                      NC    Network connections
JP     Members in the JESPlex
JC     Job classes                          CK    Health checker
SE     Scheduling environments
RES    WLM resources                        ULOG  User session log
ENC    Enclaves
PS     Processes
```

Figure 3-15 JES3 SDSF Primary Option Menu

However, it might not have all of the same options shown in the figure; the options vary according to the security level of the user. The authority to perform functions in these options also varies according to the security level of the user. It is possible to control most system functions by using the SDSF facility. The scope of the functions includes reviewing job output, controlling the processing of jobs (both their input and output), printer control, operator functions, and system administrator.

For more information, see *IBM z/OS V2R2: JES2, JES3, and SDSF*, SG24-8287.

4

z/OS maintenance concepts

Software management is a key discipline that can help you to achieve high availability (HA) and continuous operation in your IBM z/OS environment. It can also help lower the cost of installing, testing, operating, and maintaining your systems.

This chapter provides information about the following topics:

- Aspects of software management
- Software management tasks
- z/OS software management cycle
- How current your software should be

4.1 Aspects of software management

There are different aspects of software management. Although this book is focused on the z/OS platform, many of the software management concepts presented here are generally valid for other platforms, as well.

4.1.1 Why you should manage software

Managing software is important because it concerns which techniques you use to implement changes in your operating system software environment. In a typical information technology (IT) environment, it is likely that there is a continuous need for changes based on, for example, the following factors:

- Implementation of new functions
- Support for new hardware
- Software maintenance
- Implementation of new software releases

4.1.2 How current your software should be

This aspect of software management involves deciding how up-to-date you need your software to be.

4.1.3 An approach for keeping your environment current

There are different approaches and techniques for keeping your z/OS environment current. Although you can keep your system as it was when installed and only update it when errors occur, this is not a recommended approach.

Instead, you can use common tools, such as the following tools to keep your environment current:

- ServerPac and SystemPac
- Custom-Built Product Delivery Option (CBPDO)
- IBM ServicePac®

The tools are described in more detail in 2.4, “Ordering z/OS” on page 23.

4.1.4 Installation strategy

Before making a change, you must decide which upgrade method to use:

- System replacement
- System upgrade

The method you choose also depends on different considerations:

- Complexity of your environment
- Current service level
- Maturity of your system management processes (change management and problem management)
- Number of products (z/OS and vendor products)

4.1.5 Implementation strategy

Although installation is one important task, implementation is another. For example, for effective implementation, you must decide where to put data sets and how to handle IBM and vendor products, or how to prepare your system for cloning processes.

4.1.6 Concurrent maintenance

Under normal circumstances, you perform maintenance non-concurrently. Therefore, you install fixes and test them, then clone the environment and bring the actual version of the operating system or other software products into production. However, there could be situations in which it is necessary to install fixes concurrently and activate the new modules using a z/OS console command. Be aware that this method should only be used in *critical* situations.

4.2 Software management tasks

A *maintenance philosophy* involves numerous software management tasks, and it is an ongoing, cyclical process, as shown in Figure 4-1. For example, after finishing the implementation of a z/OS release, you might be asked to evaluate new functions that require program temporary fixes (PTFs).

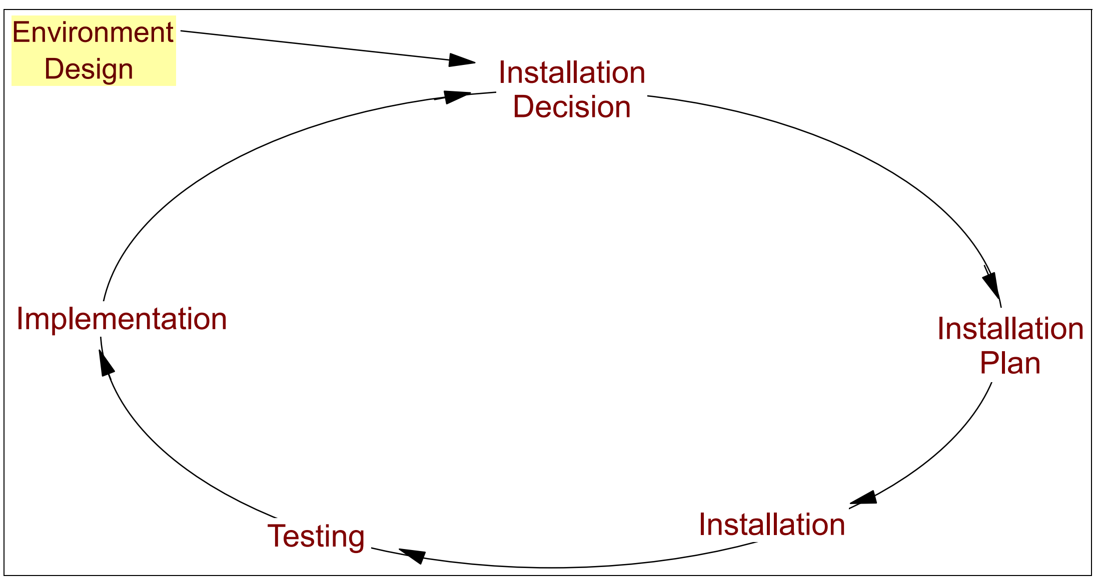

Figure 4-1 Cycle of software management tasks

4.2.1 Environment design

The environment design phase is the starting point for the cycle that follows. During this phase, you decide how to design and set up your environment for system maintenance. This phase includes the following tasks:

- Definition of installation-wide naming standards
- Logical design of the environment
- Use of shared system residences (SYSRES)
- Design of the catalog environment
- Physical design of the environment (for example, input/output (I/O) configuration)
- Cloning techniques

4.2.2 Installation decision

An installation decision is primarily a business decision, rather than a technical decision. It can involve, for example, the need for new function implementation, or the withdrawal of a z/OS release from service.

4.2.3 Installation plan

After the installation decision is made, you are responsible for the planning phase. Planning is important, because the resources you need for the subsequent installation, testing, and implementation will cost your organization money (perhaps in several areas):

- Hardware (CPC resources, DASD space)
- Software (for example, new licenses)
- Project support personnel

Planning is required for good project management. Therefore, you need to develop a plan that includes the following items:

- Key activities
- Responsibilities
- Timetable

4.2.4 Installation

In this case, installation is the process of bringing a new z/OS release to your direct access storage device (DASD) environment. As previously mentioned, you normally perform an installation by using the ServerPac dialog technique, or by using CBPDO. The ServerPac installation technique is described in 2.4, “Ordering z/OS” on page 23.

4.2.5 Testing

It is important to test new software components before introducing them into your production environment. Testing is necessary for quality assurance (QA). There are different test categories for verification and validation of a new environment. These categories can be:

- Unit tests
- Integration tests
- Function tests
- System tests
- Acceptance tests
- Regression tests
- Capacity tests
- Stress tests

4.2.6 Implementation

After completing the basic installation (for example, by using ServerPac), you still must perform many other implementation activities on your z/OS environment before you can propagate it to a test or production system.

4.3 The z/OS software management cycle

Figure 4-2 shows the software management cycle, which illustrates how software management works in a z/OS environment. From a software management perspective, it involves two major factors:

- The last software management activity you performed
- The software management strategy of your enterprise that is the base for your future activities

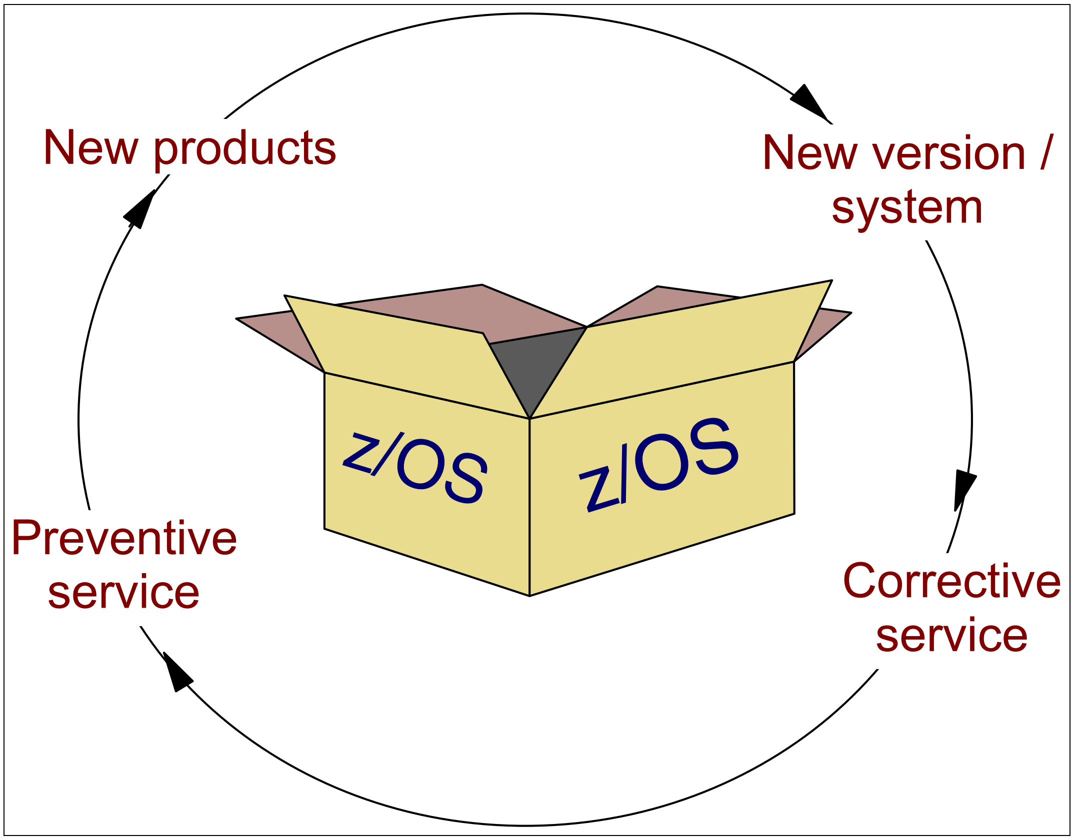

Figure 4-2 The software management cycle

4.3.1 How current your software should be

One challenge that is common to all enterprises, independent of IT infrastructure, is the risk raised by change. You might encounter a philosophy, such as "never change a running system." However, when it comes to software maintenance, an IT organization must consider the following questions:

- How current should our z/OS environment be?
- What are our guidelines and procedures for z/OS maintenance levels?
- Should we install preventive maintenance, or fix problems as they occur?

4.3.2 The risk of not changing software

There are several issues that might occur if you do not keep your z/OS software stack up-to-date:

- You might not be able to implement new software functions.
- You cannot easily implement new hardware.

- New releases of software might not interact with other software due to incompatibilities or synchronization problems.
- Problems that are already resolved by IBM might be rediscovered in your environment and might result in unnecessary outages.

4.3.3 The risk of changing software

An important part of any change is risk assessment, in which the change is considered and evaluated from the point of view of risk to the system. Low risk changes might be permitted during the day, while higher risk changes would be scheduled for an outage slot.

4.3.4 The minimum risk point

The point of minimum risk for installing new fixes to an already installed operating system (OS) release, or even to a new release, might be somewhere between being too far behind and too current.

Defining a point of minimum risk can be a challenge, so it might be helpful to first determine the requirements for your enterprise, and then compare these requirements to the level of software release that you are able to install and implement.

4.3.5 System Modification Program Extended (SMP/E)

SMP/E is the z/OS tool for managing the installation of software products on a z/OS system, and for tracking modifications to those products. SMP/E controls these changes at the component level by performing the following tasks:

- Selecting the proper levels of code to be installed from many potential changes
- Calling system utility programs to install the changes
- Keeping records of the installed changes by providing a facility to enable you to inquire on the status of your software, and to reverse the change if necessary

All code and its modifications are located in the SMP/E database called the consolidated software inventory (CSI), which consists of one or more VSAM data sets.

5

z/OS storage concepts

This chapter describes the following basic IBM z/OS storage concepts:

- Virtual storage and address spaces
- How CPC memory is managed by z/OS
- How virtual storage in managed by z/OS
- Address space map for 31-bit and 64-bit
- Dynamic address translation
- Residence Mode and Addressing Mode
- IBM Virtual Flash Memory
- Multiprogramming and multitask
- Module object and load module

For more detailed information about virtual storage concepts and IBM z/Architecture, see *ABCs of z/OS System Programming Volume 10*, SG24-6990, and *Introduction to the New Mainframe: z/OS Basics,* SG24-6366.

5.1 Processor storage overview

z/OS is known for its ability to serve thousands of users concurrently, and for processing very large workloads in a secure, reliable, and expedient manner. Its use of multiprogramming and multiprocessing, and its ability to access and manage enormous amounts of virtual and physical storage, as well as I/O operations, makes it ideally suited for running mainframe workloads.

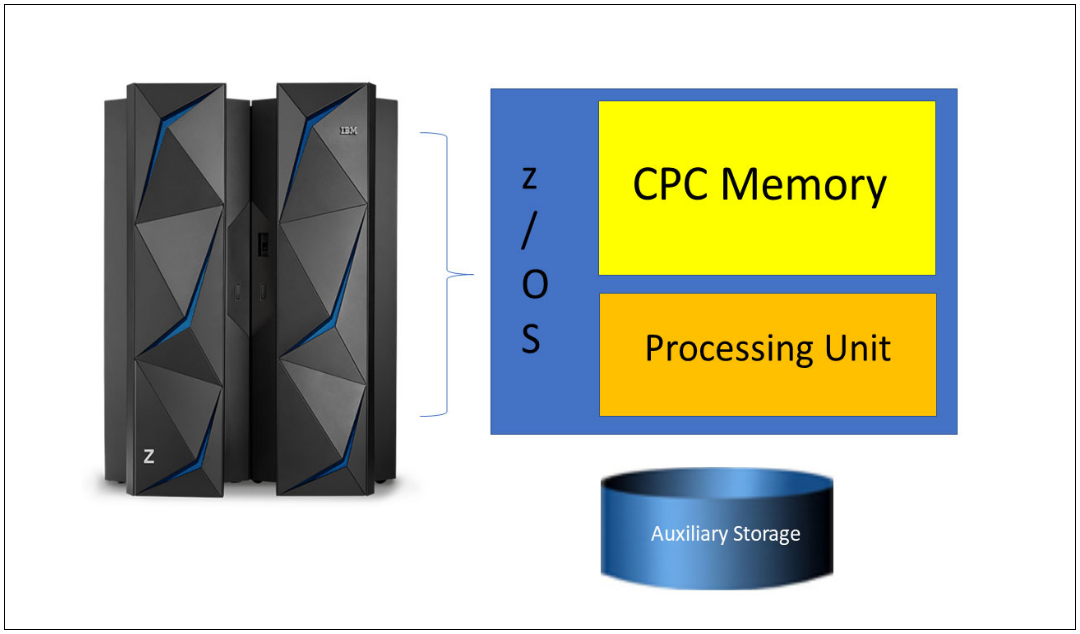

Figure 5-1 Processor storage concept

5.2 The concept of virtual storage

Virtual storage is an illusion created by the architecture, in that the system seems to have more storage than it really has. Virtual storage is created by using tables to map virtual storage pages to frames in CPC memory or slots in auxiliary storage. Only those portions of a program that are needed are actually loaded into CPC memory. z/OS keeps the inactive pieces of address spaces in auxiliary storage.

z/OS is structured around address spaces, which are ranges of addresses in virtual storage. Each user of z/OS gets an address space containing the same range of storage addresses. The use of address spaces in z/OS enables isolation of private areas in different address spaces for system security, yet also enables inter-address space sharing of programs and data through a common area accessible to every address space.

Usually the terms *CPC memory*, *processor storage*, *central storage*, *real storage*, *real memory*, and *main storage* are used interchangeably. Likewise, virtual memory and virtual storage are synonymous. The amount of CPC memory needed to support the virtual storage in an address space depends on the working set of the application being used, and this varies over time.

A user does not automatically have access to all of the virtual storage in the address space. Requests to use a range of virtual storage are checked for size limitations, and then the necessary paging table entries are constructed to create the requested virtual storage. Programs running on z/OS and IBM Z mainframes can run with 24-bit, 31-bit, or 64-bit addressing (and can switch between these modes if needed). Programs can use a mixture of instructions with 16-bit, 32-bit, or 64-bit operands, and can switch between these if needed.

5.2.1 Processor storage concept

Conceptually, mainframes and all other computers have two types of physical storage, both internal and external (see Figure 5-1 on page 68):

- Physical storage on the mainframe processor itself. This is called CPC memory, main storage, main memory, real storage, or central storage.
- Physical storage external to the mainframe, including storage on direct-access devices, such as disk drives. This storage is called paging storage or auxiliary storage.

The primary difference between the two kinds of storage relates to how they are accessed:

- CPC memory is accessed synchronously by the processor. That is, the processor must wait while data is retrieved from memory.
- Auxiliary storage is accessed asynchronously. The processor accesses auxiliary storage through an input/output (I/O) request, which is scheduled to run amid other work requests in the system. During an I/O request, the processor is free to run other, unrelated work.

CPC memory is tightly coupled with the processor itself, whereas mainframe auxiliary storage is on a (comparatively) slower, external disk. Because CPC memory is more closely integrated with the processor, it takes the processor much less time to access data from CPC memory than from auxiliary storage. However, auxiliary storage is less expensive than CPC memory.

z/OS uses both types of CPC memory and auxiliary storage to enable another kind of storage called *virtual storage*. In z/OS, each user has access to virtual storage, rather than physical memory. This use of virtual storage is central to the unique ability of z/OS to interact with large numbers of users concurrently, while processing the largest workloads.

Virtual storage means that each running program can assume it has access to all of the storage defined by the architecture's addressing scheme. The only limit is the number of bits in a storage address. This ability to use many storage locations is important because a program might be long and complex, and both the program's code and the data it requires must be in processor storage for the processor to access them.

z/OS supports 64-bit long addresses, which enables a program to address up to 18,446,744,073,709,600,000 bytes (16 exabytes) of storage locations. In reality, the mainframe has much less CPC memory installed.

To enable each user to act as though this much storage really exists in the computer system, z/OS keeps only the active portions of each program in CPC memory. It keeps the rest of the code and data in files called *page data sets* on auxiliary storage, which usually consists of a number of high-speed direct access storage devices (DASD). In general, when you refer to a file in z/OS, you refer to a *data set*.

Virtual storage, then, is this combination of real and auxiliary storage. z/OS uses a series of tables and indexes to relate locations on auxiliary storage to locations in CPC memory. z/OS uses various storage manager components to manage virtual storage.

Mainframe users use the terms CPC memory, processor storage, central storage, real memory, real storage, and main storage interchangeably. Likewise, they use the terms *virtual memory* and *virtual storage* synonymously.

5.2.2 How virtual storage works

By bringing pieces of the program into CPC memory only when the processor is ready to run them, z/OS can run more and larger programs concurrently.

For a processor to run a program instruction, both the instruction and the data it references must be in CPC memory. The entire program does not really need to be in CPC memory when an instruction runs. Rather, by bringing pieces of the program into CPC memory only when the processor is ready to run them, moving them out to auxiliary storage when it doesn't need them, an operating system can run more and larger programs concurrently.

How does it work? Physical storage is divided into areas, each the same size, and accessible by a unique address. In CPC memory, these areas are called *frames*; in auxiliary storage, they are called *slots*. Similarly, the operating system can divide a program into pieces the size of frames or slots, and assign each piece a unique address. This arrangement allows the operating system to track these pieces. In z/OS, the program pieces are called pages.

Pages are referenced by their virtual addresses and not by their real addresses. From the time that a program enters the system until it completes, the virtual address of the page remains the same, regardless of whether the page is in CPC memory or auxiliary storage. Each page consists of individual locations called bytes, each of which has a unique virtual address.

Within z/Architecture, you can define three page sizes: 4 KB (the most common one), 1 MB, and 2 GB.

5.2.3 Dynamic address translation

Dynamic address translation (DAT) is the process of translating a virtual address during a storage reference into the corresponding real address.

If the virtual address is already in CPC memory, the DAT process can be accelerated by using a translation lookaside buffer. If the virtual address is not in CPC memory, a page fault interrupt occurs, z/OS is notified and brings the page in from auxiliary storage.

DAT is implemented by both hardware and software by using page tables, segment tables, region tables, and translation lookaside buffers. DAT allows different address spaces to share program or other data that is for read only. This is because virtual addresses in different address spaces can be made to translate to the same frame of CPC memory. Otherwise, there would have to be many copies of the program or data, one for each address space.

5.2.4 Address space

The range of virtual addresses that the operating system assigns to a user or separately running program is called an address space (see “The address space concept” on page 78). This is the area of contiguous virtual addresses available for running instructions and storing data. The range of virtual addresses in an address space starts at zero and can extend to the highest address permitted by the operating system architecture.

z/OS provides each user with a unique address space and maintains the distinction between the programs and data belonging to each address space. In some ways, a z/OS address space is like a UNIX process, and the address space identifier (ASID) is like a process ID (PID).

However, the use of multiple virtual address spaces in z/OS holds some special advantages. Virtual addressing permits an addressing range that is greater than the CPC memory capabilities of the system. The use of multiple virtual address spaces provides this virtual addressing capability to each job in the system by assigning each job its own separate virtual address space. The potentially large number of address spaces provides the system with a large virtual addressing capacity.

With multiple virtual address spaces, errors are confined to one address space, except for errors in commonly addressable storage, thus improving system reliability and making error recovery easier. Programs in separate address spaces are protected from each other. Isolating data in its own address space also protects the data.

z/OS uses many address spaces. There is at least one address space for each job in progress and one address space for each user logged on through TSO/E. There are many address spaces for operating system functions, such as operator communication, automation, networking, security, and so on.

The use of address spaces allows z/OS to maintain the distinction between the programs and data belonging to each address space. The private areas in one user's address space are isolated from the private areas in other address spaces, and this address space isolation provides much of the operating system's security.

Yet, each address space also contains a common area that is accessible to every other address space. Because it maps all of the available addresses, an address space includes both system code and data, and user code and data. Therefore, not all of the mapped addresses are available for user code and data.

The ability of many users to share the same resources implies the need to protect users from one another, and to protect the operating system itself. Along with such methods as "keys" for protecting CPC memory and code words for protecting data files and programs, separate address spaces ensure that users' programs and data do not overlap.

5.2.5 Storage initialization

The largest CPC memory available is 32 terabytes (32 TB) for the IBM z14.

The system initialization process begins when the system operator selects the LOAD function at the system console. This causes an initial program load (IPL), which is equivalent to a boot in other platforms. z/OS locates all of the usable CPC memory that is online and available in the logical partition (LPAR) that has undergone IPL, creating a virtual storage environment for the building of various system areas. z/OS uses CPC memory to map the virtual storage, which implies allocating and using auxiliary storage.

5.2.6 CPC memory

CPC memory (sometimes referred to as *main storage*), provides the system with a volatile processor that is directly addressable, with fast access for the electronic storage of data.

A portion of the CPC memory can be defined as Virtual Flash Memory (VFM) used by z/OS to handle picks of paging process (see 5.6, "IBM Virtual Flash Memory" on page 82).

Both data and programs must be loaded into CPC memory from DASD devices (or *moved* from VFM) before they can be processed by the processors. The maximum CPC memory size per LPAR is restricted by hardware, and the z/OS and system architecture. For the IBM z14, the maximum LPAR size is 16 TB; z/OS V2.3 supports up to 4 TB.

5.2.7 Auxiliary storage

Auxiliary storage DASD hard disk drives are required on z/OS systems for storing z/OS page data sets. For additional paging flexibility and efficiency, you can add optional storage-class memory (SCM) on Virtual Flash Memory as a second type of auxiliary storage (see 5.6, “IBM Virtual Flash Memory” on page 82).

You must have enough auxiliary storage available to store the programs and data that comprise your z/OS system.

Page data sets are used to implement virtual storage, which contains the paged-out portions of all virtual storage address spaces.

5.2.8 Processing unit

Figure 5-1 on page 68 depicts a processing unit (PU). The PU is the hardware in charge of running instructions located in CPC memory. The PU contains the sequencing and processing facilities for instruction execution, interruption action, timing functions, IPLs, and other machine-related functions. PUs and CPC memory are packed in units known as CPC drawers in the A-Frame of the IBM z14.

5.3 Virtual storage details

As mentioned before, virtual storage is an illusion created by the architecture, in that the system seems to have more storage than it really has. Each user or program gets an address space, and each address space contains the same range of storage addresses. Only those portions of the address space that are needed at any point in time are actually loaded into CPC memory. z/OS keeps the inactive pieces of address spaces in auxiliary storage.

Figure 5-2 depicts virtual storage used as auxiliary storage.

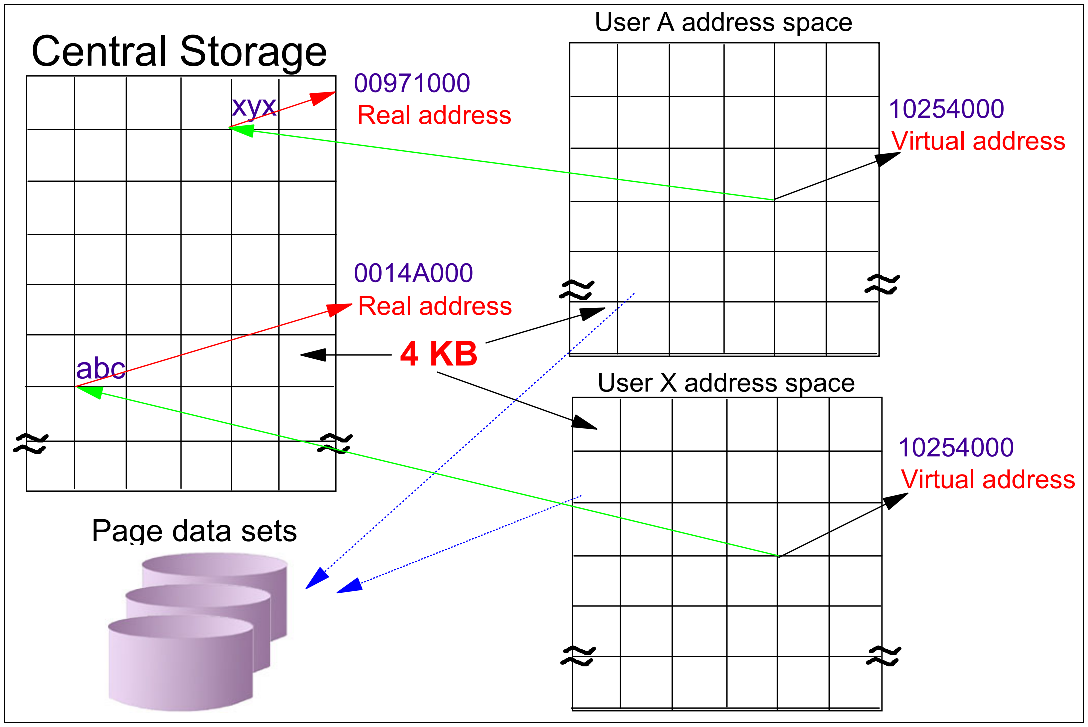

Figure 5-2 Virtual storage, auxiliary storage

z/OS manages address spaces in units of various sizes:

- Four kilobyte (4 KB) units of virtual storage (pages). z/OS also supports 1 MB and 2 GB pages called *large pages*. These instances are special cases; for more information, see “1 MB and 2 GB Large Pages” on page 93.
- One megabyte (1 MB) units called *segments*. A segment is a block of sequential virtual addresses spanning megabytes, beginning at a 1 MB boundary. A 2-gigabyte (GB) address space, for example, consists of 2048 segments.
- A virtual address, accordingly, is divided into four principal fields:
  - Bits 0-32 are called the region index (RX).
  - Bits 33-43 are called the segment index (SX).
  - Bits 44-51 are called the page index (PX).
  - Bits 52-63 are called the byte index (BX).
- An address (called virtual) as referred to by a program is an identifier of a required piece of information in CPC memory. This enables the size of an address space (all virtual addresses available to a program) to exceed the CPC memory size.
- All CPC memory references are made in terms of virtual storage address.
- DAT is used to perform a mapping between the virtual storage address and its physical location in CPC memory.

5.3.1 Virtual storage terminology

Putting all the pieces together we can say that in z/OS, virtual storage is implemented (transparently to the program) using the following concepts:

Segment Program address space is divided into segments of 1 MB addresses in size.

Page A segment is divided into pages, which are blocks of 4 KB addresses in size.

Frame CPC memory is divided into frames, which are blocks of 4 KB in size.

Slots Auxiliary storage page data sets are formatted in slots, which are blocks of 4 KB in size.

DAT DAT is implemented by hardware and by software throughout the use of tables defined by z/Architecture.

A z/OS program accesses addresses located in virtual storage. Only pages of the program currently active need to be in a CPC memory frame at processing time. The inactive pages are held in auxiliary storage.

Temporary data sets (work files) can be handled by a function called virtual I/O (VIO). Data sets for which VIO is specified are located in external page storage. However, to the program, the data sets appear to be on real direct access storage devices.

5.3.2 Page data sets

Page data sets contain the paged-out portions of all virtual storage address spaces. In addition, output to VIO data sets can be stored in the page data sets. The installation must allocate sufficient space on page data sets to back up all of the virtual address spaces being run concurrently by the processor, plus the space required for VIO data sets.

Frames, slots, and pages

When a program is selected, z/OS brings it into virtual storage (enabling it to use a range of virtual addresses) and divides it into pages of 4000 addresses. Then, z/OS transfers the pages of the program into CPC memory for execution, and out to auxiliary storage when CPC memory is under contention.

Not all pages of a program are necessarily in CPC memory at one time. To the programmer, the entire program appears to occupy a contiguous space of addresses in CPC memory at all times.

However, the pages that are in CPC memory do not necessarily occupy contiguous space. Figure 5-3 shows this.

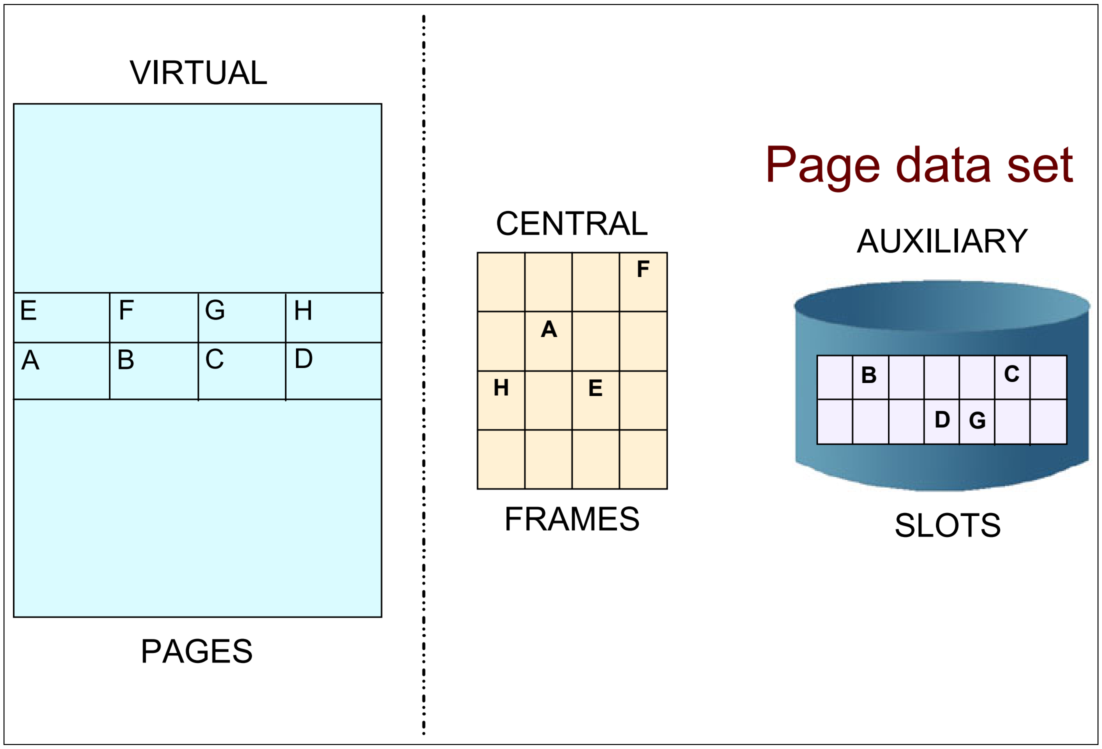

Figure 5-3 Storage frames, slots, and pages

5.3.3 Pages to auxiliary storage (paging)

To understand how paging works, assume that DAT encounters an invalid page table entry during address translation, indicating that a page is required that is not in a CPC memory frame. To resolve this page fault, the system must bring the page in from auxiliary storage.

However, it must first locate an available CPC memory frame. If none is available, the request must be saved and an assigned frame freed. To free a frame, the system copies its contents to auxiliary storage, and marks its corresponding page table entry as invalid. This operation is called a page-out.

The parts of a program running in virtual storage must be moved between real and auxiliary storage. To enable this, z/OS breaks the storage into blocks of 4096 bytes (4 KB):

- A block of 4 KB of CPC memory is a *frame*.
- A block of 4 KB addresses in virtual storage is a *page*. A virtual storage page is backed by:
  - CPC memory
  - Auxiliary storage
- A block of storage on an auxiliary device is a *slot*.

Frames, pages, and slots are all the same size (4 KB). An active virtual storage page resides in a CPC memory frame. A virtual storage page that becomes inactive resides in an auxiliary storage slot (in a page data set). Figure 5-3 shows the relationship of pages, frames, and slots.

In Figure 5-3 on page 75, z/OS is performing paging for a program running in virtual storage. The lettered boxes represent parts of the program. In this simplified view, program parts A, E, F, and H are active and located in CPC memory frames. Program parts B, C, D, and G are inactive and have been moved to auxiliary storage slots. However, all of the program parts reside in virtual storage and have virtual storage addresses.

5.3.4 Storage management

z/OS tries to keep an adequate supply of available CPC memory frames on hand. When a program refers to a page that is not in CPC memory, z/OS uses a CPC memory frame from a supply of available frames and reads in the program page from the corresponding page data set. See section 5.2.7, “Auxiliary storage” on page 72.

When this supply becomes low, z/OS uses a page stealing algorithm to replenish it. It takes a frame assigned to an active user and makes it available for other work. The decision to “steal” a particular page is based on the activity history of each page currently residing in a CPC memory frame. Pages that have not been active for a relatively long time are good candidates for page stealing.

When the program running in the address space is not doing any productive work (for example, a TSO/E user is inputting a command), z/OS removes the address space from the CPC memory and place it into auxiliary storage. This is called *swapping*. Swapping is the process of transferring all of the pages of an address space between central storage and auxiliary storage.

A swapped-in address space is active, having pages in CPC memory frames and pages in auxiliary storage slots. A swapped-out address space is inactive; the address space resides on auxiliary storage and cannot run until it is swapped in.

5.3.5 Paging algorithms

z/OS uses a sophisticated paging algorithm to efficiently manage virtual storage based on which pages were most recently used. An unreferenced interval count indicates how long it has been since a program referenced the page. For real storage management purposes, each frame in the CPC memory has two bits associated: one is called the *reference bit* and the other is called the *change bit*.

At regular intervals, the system checks the reference bit for each page frame. If the reference bit is off (the frame has not been referenced), the system adds to the frame’s unreferenced interval count. It adds the number of seconds since this address space last had the reference count checked.

If the reference bit is on, the frame has been referenced, and the system turns it off and sets the unreferenced interval count for the frame to zero. Frames with the highest unreferenced interval counts are the ones most likely to be stolen.

If the frame is stolen, z/OS checks the change bit; the change bit is turned on by the processor if and area in the page has been altered. If that is the case, the page is sent to the page data set. If the page was not changed, there is no need to send it to the page data set because there is a valid copy of the page there.

5.4 z/Architecture address space

As shown in Figure 5-4, the 2 GB address in the address space is marked by a virtual line called *the bar*. The bar separates storage below the 2 GB address, called *below the bar* from storage above the 2 GB address, called *above the bar*. The area above the bar can be used for data and for running programs.

Your installation can set a limit on the use of the address space above the bar for a single address space.

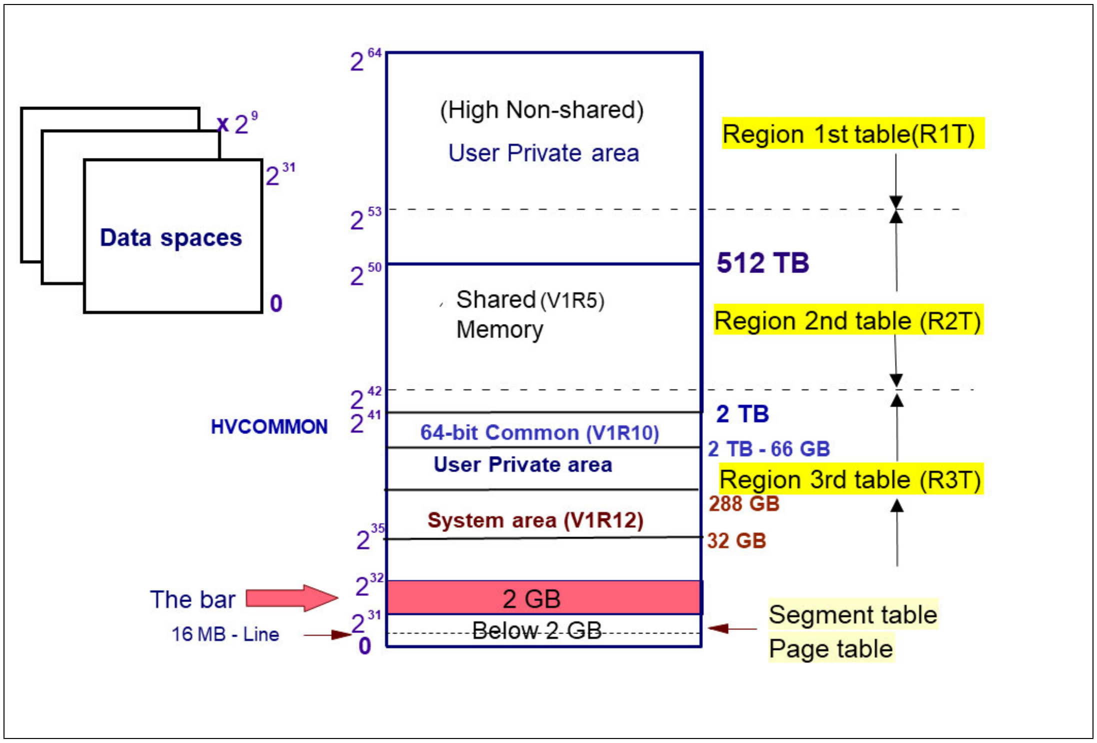

Figure 5-4 z/Architecture address space

5.4.1 Virtual storage and 64-bit addressability

A program running in an address space can reference all of the storage associated with that address space. A program's ability to reference all of the storage associated with an address space is called *addressability*.

5.4.2 zSeries mainframes and 64-bit addressing

With 64-bit addressing, the potential size of a z/OS address space expands to a size so vast that you need new terms to describe it, as shown in Figure 5-5 on page 78.

Each address space, called a *64-bit address space*, is 16 EB. An *exabyte* is slightly more than one billion gigabytes. The 64-bit address space is 8 billion times the size of the former 2 GB address space, or 18,446,744,073,709,600,000 bytes.

A program running on z/OS can run with 24-bit, 31-bit, or 64-bit addressing mode (and can switch if needed). To address the high virtual storage available with the 64-bit architecture, the program uses 64-bit-specific instructions. Although the architecture introduces unique 64-bit exploitation instructions, the program can use both 31-bit and 64-bit instructions, as needed.

5.4.3 Region tables and segment tables

In a 16 EB address space with 64-bit virtual storage addressing, the following levels of translation tables are defined by the architecture:

- Region third table (R3T)
- Region second table (R2T)
- Region first table (R1T)
- Segment table
- Page table

The region tables are 16 KB in length, and there are 2048 entries per table. Each entry in the R3T table points to a segment table. The segment tables also have 2048 entries; the size of a segment table is also 16 KB. See “64-bit virtual address” on page 92 for details about how the translation process is performed.

5.4.4 The address space concept

As defined in “Address space” on page 70, the range of virtual addresses that the operating system assigns to a user or separately running program is called an *address space*. This is the area of contiguous virtual addresses available for running instructions and storing data. The range of virtual addresses in an address space starts at zero and can extend to the highest address permitted by the architecture.

As shown in Figure 5-5, the area above the 2 GB address is called the *bar*. The addresses above the bar can also be used for programs and data.

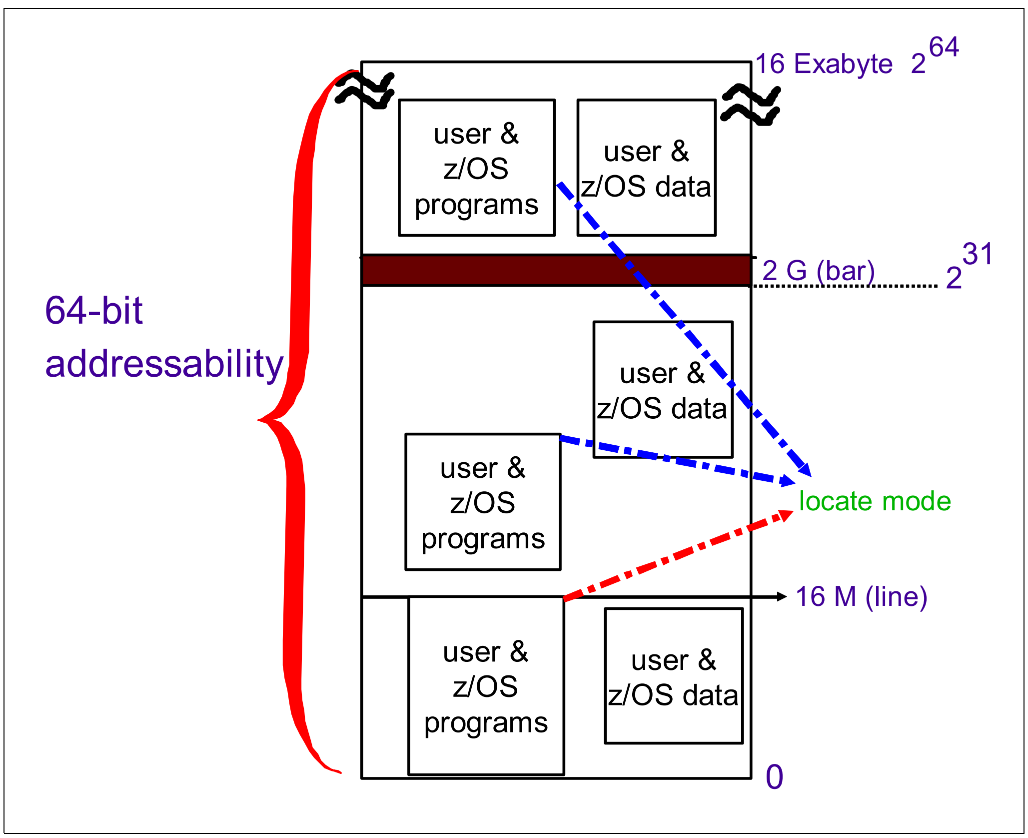

Figure 5-5 Address space concept

5.4.5 Data spaces

A *data space* is a type of virtual storage space with a range up to 2 GB of contiguous virtual storage. The virtual storage map of a data space is quite different from an address space. The entire 2 GB is available for user data, and does not contain specific areas.

A data space can hold only data; it does not contain z/OS control blocks or programs in execution. Program code does not run in a data space, although a program can reside in a data space as data (however, to be run it needs to be copied to an address space). A program can refer to data in a data space at bit level, as it does in a work file.

5.4.6 Addressing mode and residency mode

Every program that runs in z/OS is assigned two attributes, an AMODE (addressing mode) and an RMODE (residency mode). Figure 5-6 depicts the AMODE and RMODE attributes.

AMODE is a program attribute to indicate which hardware addressing mode should be active to solve an address; that is, how many bits are to be used for solving and dealing with addresses:

- `AMODE=24` indicates that the program can address up to 16 M virtual addresses.
- `AMODE=31` indicates that the program can address up to 2 G virtual addresses.
- `AMODE=64` indicates that the program can address up to 16 EB virtual addresses (only in z/Architecture).

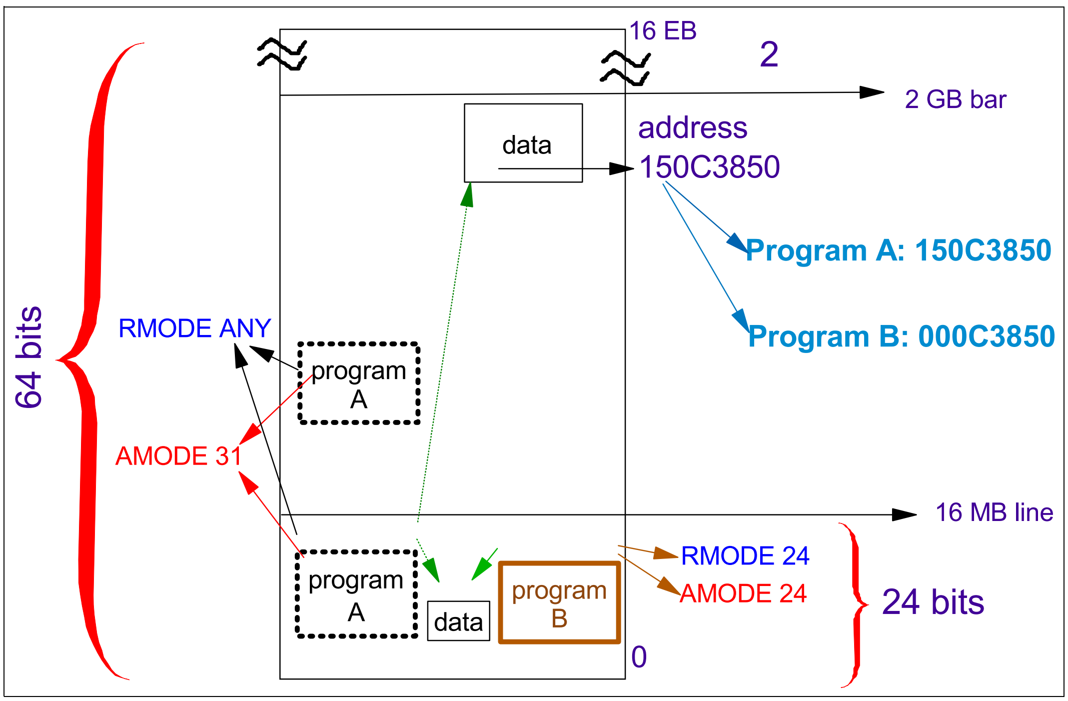

Figure 5-6 Addressing mode and residency mode

The concept of *residency mode* (RMODE) is used to indicate where a program is to be placed in the virtual storage (by z/OS program management) when the system loads it from DASD:

- `RMODE=24` indicates that the module *must* reside below the 16 MB virtual storage line. Among the reasons for RMODE24 are that the program is AMODE24, the program has control blocks that must reside below the line.

- RMODE=ANY indicates that the module can reside anywhere in virtual storage, but preferentially above the 16 MB virtual storage line. Because of this, such an RMODE is also called RMODE 31.
- RMODE=64 Indicates that the module might reside in virtual storage either above or below the 2 GB virtual storage bar.

5.5 Storage managers

In a z/OS system, storage is managed by the z/OS components virtual, real, and auxiliary storage managers, shown in Figure 5-7.

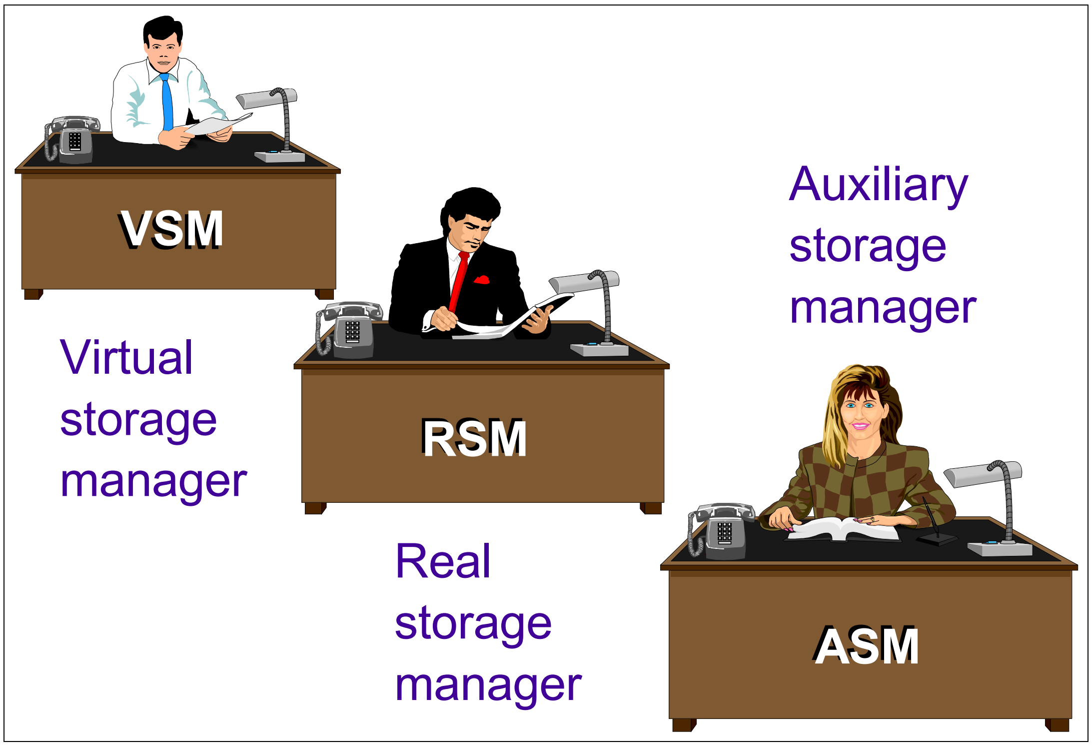

Figure 5-7 The storage component managers

The following list describes the various managers:

- Virtual storage manager (VSM): The z/OS component that manages virtual storage. Its main function is to control the use of virtual storage addresses. Each installation can use virtual storage parameters to specify how certain virtual storage areas are to be allocated to programs. These parameters have an effect on CPC memory use and overall system performance. VSM keeps track of the map of virtual storage for each address space.
- Real storage manager (RSM): The z/OS component that controls the usage of CPC memory frames. RSM acts together with ASM to support the virtual storage concept, and with VSM to ensure that a page is backed up in a CPC memory frame. Furthermore, RSM establishes many services to other components and application programs to manipulate the status of pages and frames.

  RSM keeps track of the contents of CPC memory. It manages the paging activities, such as page-in, page-out, page stealing, and helps with swapping an address space in or out. RSM also performs page fixing, which is marking pages as unavailable for stealing.

- Auxiliary storage manager (ASM): ASM is a z/OS component responsible for transferring virtual pages between processor frames and auxiliary storage slots (page data sets). This is done as either a paging operation (one page at time), or as a physical swapping operation (an address space, all pages at a time). ASM manages the transfer by initiating the I/O and by maintaining tables to reflect the current status of the slots. This status includes the location of each page in each slots.

  ASM controls the use of page data sets and the implicit paging I/O operation. As a system programmer, you are responsible for the size and the performance of the page data sets. The ASM uses the system's page data sets to keep track of auxiliary storage slots:

  – Slots for virtual storage pages that are not in CPC memory frames
  – Slots for pages that do not occupy frames but, because the frame's contents have not been changed, the slots are still valid

  When a page-in or page-out is required, ASM works with RSM to locate the proper CPC memory frames and auxiliary storage slots. It builds segment and page tables that are used to translate a virtual address to a real address.

  ASM attempts to maximize page I/O efficiency by incorporating a set of algorithms to distribute the I/O load evenly (through the local page data sets). In addition, every effort is made to keep the system operable in situations where a shortage of a specific type of slots exists. ASM selects a local page data set for page-out from its available page data sets. ASM selects these data sets in a circular order in each type of data set, subject to the availability of free space and the device response time.

5.5.1 Paging and swapping

Paging is the movement of pages between CPC memory frames and auxiliary storage slots. There are two types of paging operations:

- *Page-in*, which flows from a slot to a frame. It is caused by a page fault. A page fault is an interrupt caused by the hardware in charge of translating a virtual address into a real address. The page fault happens because the page is not currently mapped to a frame. RSM gains control and, through ASM, provides a page-in operation to retrieve the page from auxiliary storage.
- *Page-out*, which flows from a frame to a slot. It is caused when a changed page needs to be stolen from CPC memory because this memory is under contention. RSM calls ASM to schedule the paging I/O necessary to send these pages to auxiliary storage.

As mentioned, swapping is the process of transferring all of the pages of an address space between CPC memory and auxiliary storage. A swapped-in address space is active, having pages in CPC memory frames and pages in auxiliary storage slots. A swapped-out address space is inactive. The address space resides on auxiliary storage and cannot run until it is swapped in. z/OS implements the concept of a logical swap. When z/OS decides to swap an address space it is removed from the processor execution queues but the recently referenced frames are kept in CPC memory.

Auxiliary page data sets are formatted in slots. They should contain pages that for various reasons are not to stay in CPC memory frames. Contention is reduced when these classes of pages are placed on different physical devices. Multiple page data sets are preferable.

5.6 IBM Virtual Flash Memory

The IBM Virtual Flash Memory (VFM) is the next generation of storage class memory designed to help improve availability and performance during workload transitions for improved quality of service. Virtual Flash Memory can help reduce latency for critical paging that might otherwise impact the availability and performance of your key workloads.

Figure 5-8 shows IBM z14 memory, consisting of addressable memory (VFM, customer ordered memory, and the hardware system area or HSA) and installed physical memory, which includes the redundant array of independent memory (RAIM) with DIMM capacity.

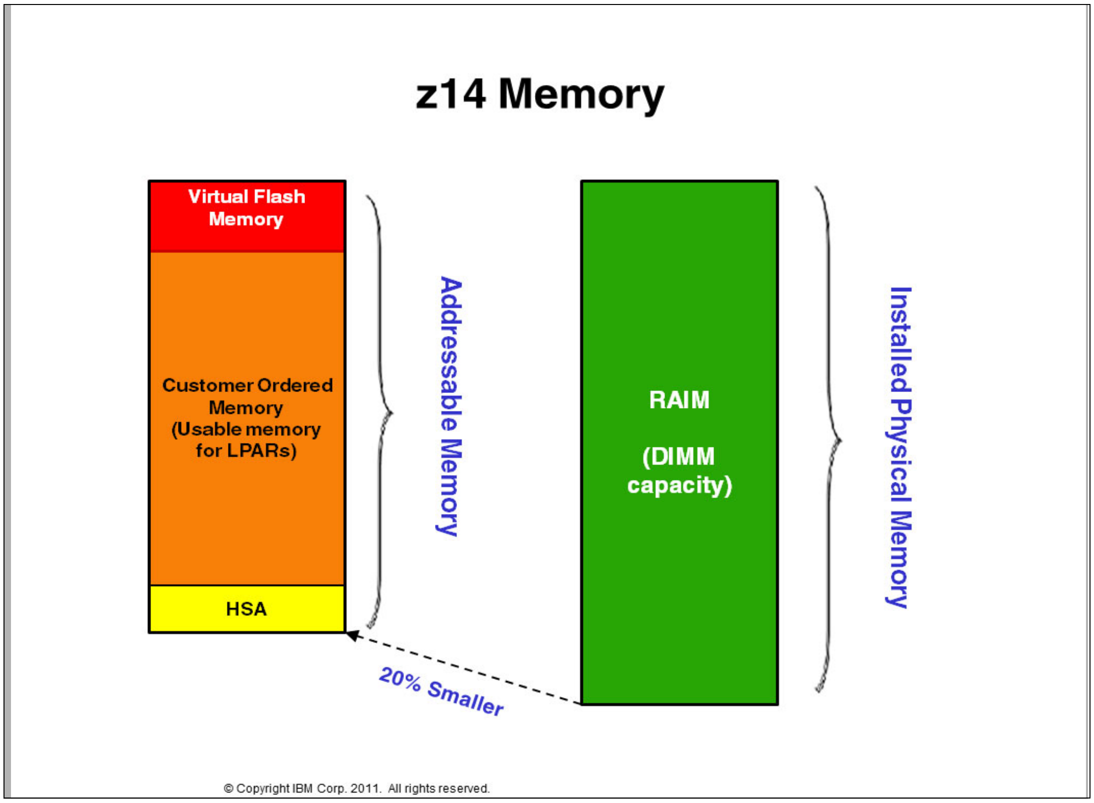

Figure 5-8 IBM Virtual Flash Memory

Virtual Flash Memory is designed to offer exceptional performance for paging spikes by reducing paging latency. Virtual Flash Memory can be especially helpful during transitional workload processing shifts where paging might surge, such as during the start of the day, changes in loads, or diagnostic data collection.

VFM is designed to help improve availability and handling of paging workload spikes when running z/OS V2.1, V2.2, or V2.3. With this support, z/OS is designed to help improve processor performance by supporting middleware exploitation of pageable large (1 MB) pages. VFM can also be used in coupling facility images to provide extended capacity and availability for workloads that use WebSphere MQ Shared Queues structures.

Using VFM can help availability by reducing latency from paging delays that can occur at the start of the workday or during other transitional periods. It is also designed to eliminate delays that can occur when collecting diagnostic data during failures. Therefore, VFM can help meet most demanding service level agreements and compete more effectively. VFM is easy to configure and provides rapid time-to-value.

No application changes are required to migrate from IBM Flash Express to VFM to help improve system availability and responsiveness by using VFM across transitional workload events such as market openings.

5.6.1 Storage-class memory on IBM Z Virtual Flash Memory

The storage provided by VFM is called storage-class memory (SCM), as shown in Figure 5-9. When enabled, SCM is used by z/OS for paging (4K and larger pages), and for staging of IBM System Storage® SAN Volume Controller (SVC) dumps.

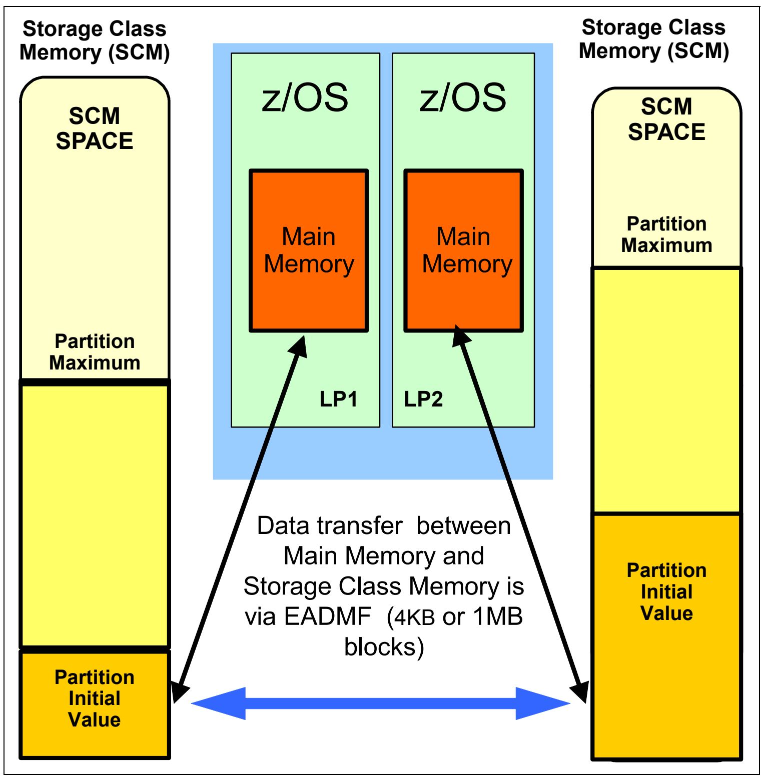

Figure 5-9 Paging to Storage Class Memory

You can define the initial and maximum amount of Virtual Flash Memory processor partition, called LPAR, that z/OS will be running on (for more details about LPARs, see *ABCs of z/OS System Programming Volume 10*, SG24-6990). The maximum memory that is allocated to an LPAR can be dynamically changed. On z/OS, this process can also be done by using an operator command.

Virtual Flash Express is used by the ASM with paging data sets to satisfy page-out and page-in requests received from the RSM. It supports 4 KB and 1 MB page sizes. ASM determines where to write a page based on space availability, data characteristics, and performance metrics.

5.7 The common virtual storage area

The z/OS implementation of virtual storage is to have one address space per set of related programs. The advantage of this design is isolation; any error is contained in one address space and cannot be propagated to another address space. Also, because the number of address spaces can be large, the number of virtual addresses to be used by programs is enormous.

However, such a design poses a problem: The need for communication between programs from different address spaces. To solve that problem, the common area was introduced. All address spaces in a z/OS system image share a virtual storage area known as the *common area* (see Figure 5-10). That means that all address spaces programs in this z/OS access the same common data and the same common programs, with the same virtual address.

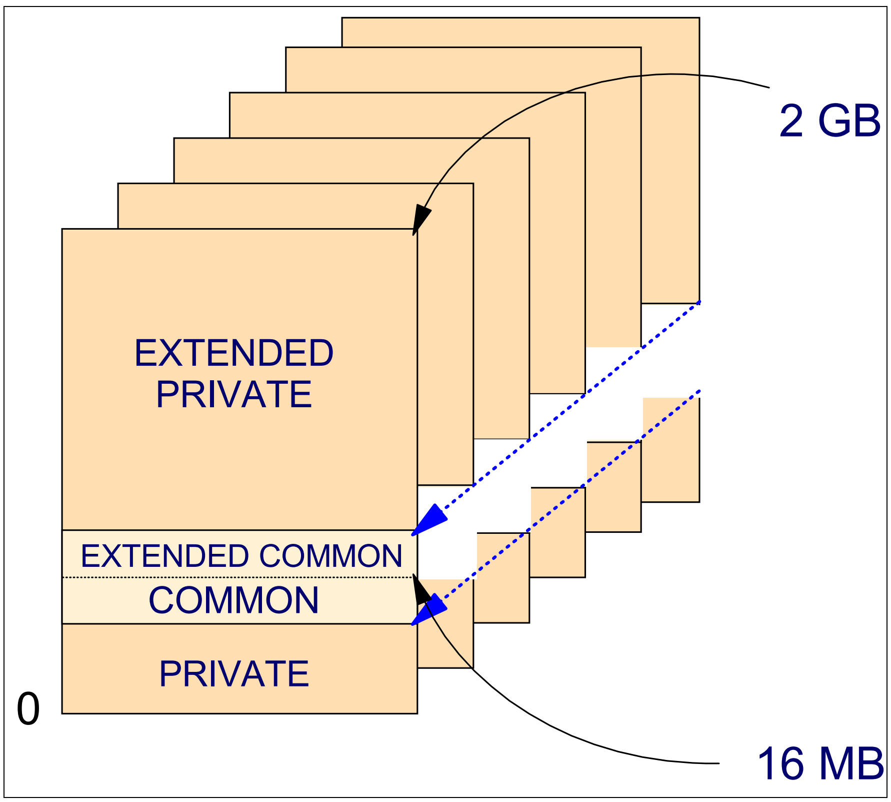

Figure 5-10 Common storage area

5.7.1 Common area below the 16 MB line

Each storage area in the common area (below 16 MB) has a counterpart in the extended common area (above 16 MB), except the prefixed save area (PSA). The common area contains system control programs and control blocks.

The following storage areas are located in the common area:

- Prefixed storage area

  This area is often referred to as *low core*. The PSA is a common area of virtual storage from address zero through 8191 in every address space. There is one unique PSA for every processor installed in a system.

The PSA maps architecturally fixed hardware and software storage locations for the processor. Because there is a unique PSA for each processor, from the view of a program running on z/OS, the contents of the PSA can change any time that the program is dispatched on a different processor. This feature is unique to the PSA area, and is accomplished through a unique DAT manipulation technique called *prefixing*.

- Common service area

  This portion of common area storage (addressable by all address spaces) is available to all applications. The CSA is often used to contain data frequently accessed by multiple address spaces. The size of the CSA area is established at system initialization time (IPL), and cannot change when the operating system is active.

- Pageable link pack area, fixed link pack area, and modified link pack area

  This area contains the link pack areas, which are the PLPA, fixed link pack area (FLPA), and modified link pack area (MLPA), contain system-level programs that are often run by multiple address spaces. For this reason, the link pack areas reside in the common area, which is addressable by every address space, eliminating the need for each address space to have its own copy of the program. This storage area is below the 16 MB boundary.

- System queue area

  This area contains system-level data accessed by multiple address spaces. The SQA area is not pageable (fixed), which means that it resides in CPC memory until it is freed by the requesting program. The size of the SQA area is predefined by the installation, and cannot change when the OS is active. Yet it has the unique ability to “overflow” into the CSA area if there is unused CSA storage that can be converted to SQA.

- Nucleus, which is fixed and non-swappable

  This is a read-only area of common storage that contains z/OS control programs.

5.7.2 z/OS nucleus

The nucleus in the common area contains the z/OS nucleus programs (kernel) and extensions to the nucleus that are initialized during IPL processing. The nucleus contains the most important z/OS programs. The nucleus RMODE24 programs reside below the 16 MB line. The nucleus RMODE31 programs reside above the 16 MB line.

5.7.3 System queue area

The system queue area (SQA) is a common area containing control blocks used by z/OS to manage transaction workloads and the use of system resources. The number of active address spaces (which depends on the workload run in the system) affects the system’s use of SQA.

SQA is allocated directly below the nucleus, as shown in Figure 5-11. Extended SQA (ESQA) is allocated directly above the extended nucleus. Both allocations occur at IPL time. The size of SQA is defined by the installation.

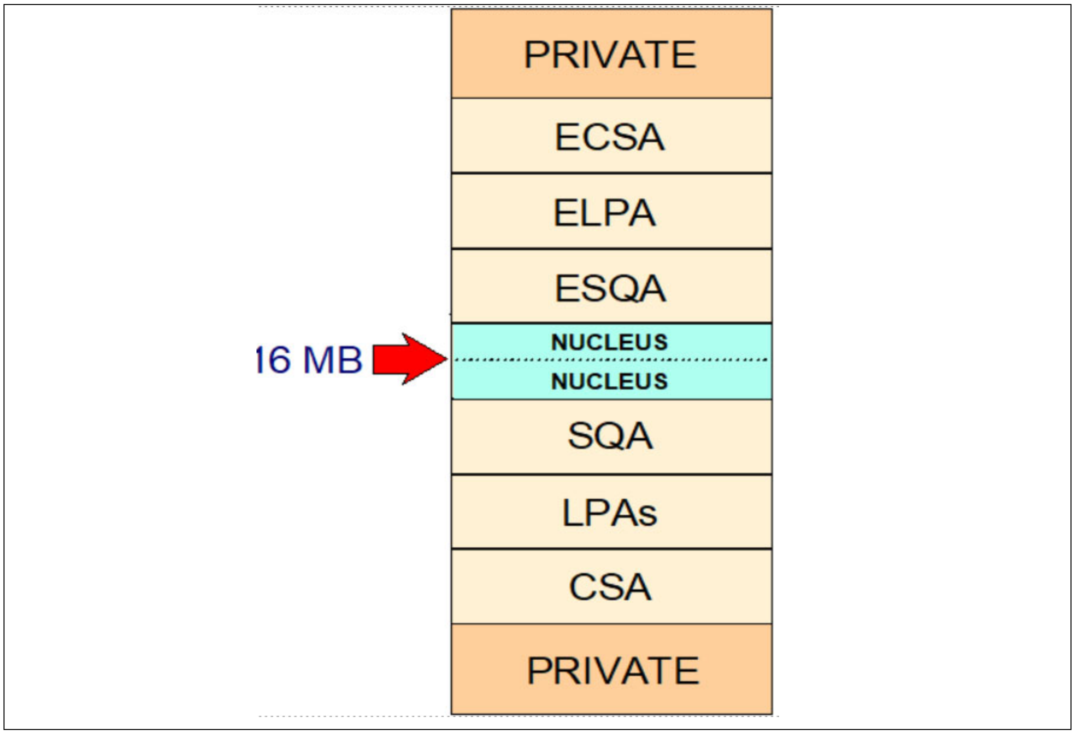

Figure 5-11 z/OS Address Space layout

5.7.4 Common service area

The CSA is a common area containing control blocks used by subsystem programs, such as JES2, Data Facility Storage Management Subsystem (DFSMS), and IBM Resource Access Control Facility (RACF), and access methods, such as Virtual Storage Access Method (VSAM).

CSA/ECSA normally contains data referenced by several system address spaces, enabling address spaces to communicate by referencing the same piece of CSA data. In a sense, CSA/ECSA looks like SQA/ESQA.

CSA is allocated directly below the MLPA. ECSA is allocated directly above the extended MLPA, as shown in Figure 5-11. If the virtual SQA/ESQA space is full, z/OS allocates more SQA/ESQA space from the CSA/ECSA.

5.7.5 Link pack area

The LPA and ELPA contain programs that are preinstalled at IPL time in the common area, from the z/OS program data sets. These programs are z/OS and installation defined read-only programs (the ones not modified during its execution).

Because such code is in the common area, all of these single-copy programs can be run in any address space. Their copy is not self-modifying (reentrant), so the same copy of the program can be used by any number of address spaces at the same time. This reduces the demand for CPC memory. The RMODE attribute of the program decides its location (LPA or ELPA). The ELPA is built above 16 MB.

5.7.6 Address space private area

The portion of the user's private area in each virtual address space that is available to the user's programs is called the *user region*. The use of CPC memory address spaces enables z/OS to maintain the distinction between the programs and data belonging to each address space. The private areas in one user's address space are isolated from the private areas in other address spaces, and this address space isolation provides much of the z/OS's security.

There are two private areas: below the 16 MB line is the private (PVT), and above the 16 MB line is the extended private (EPVT). Their size is the complement of the common area's size. The virtual addresses in the private area are unique to the programs running in such areas.

5.8 64-bit address space map

As previously mentioned, z/Architecture broke the 2 GB (31-bit) main storage limit and the 2 GB (31-bit) address limit, and moved the limit to 16 EB (64-bit). The maximum of a z/OS address space is 16 EB addresses, which makes the new address space 8 billion times the size of the former 2 GB address space. However, any new created address space in z/OS is initialized with 2 GB addresses with the potential to go beyond.

For compatibility, the layout of the virtual storage areas for an address space is the same under 2 GB. The area that separates the virtual storage area below the 2 GB address from the user private area is called the *bar*, as shown in Figure 5-12, and is 2 GB addresses thick. In a 64-bit virtual storage environment, the terms are used to identify the areas *above the bar* (2**32 - 2**64-1) and *below the bar* (0 - 2**31-1).

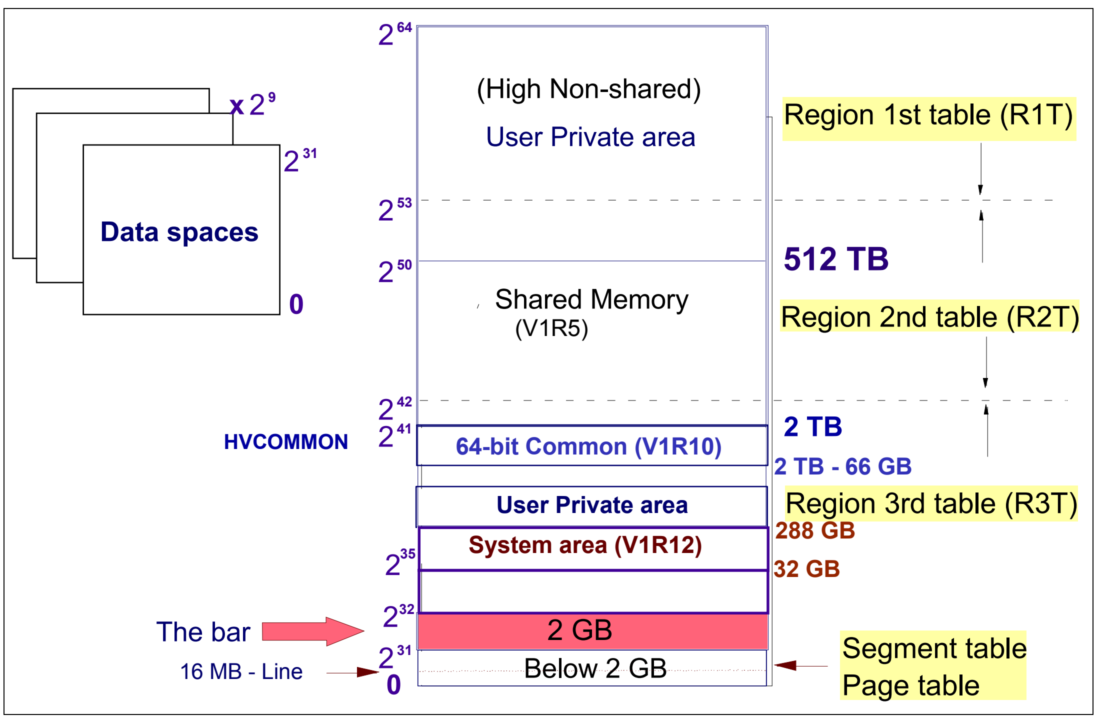

Figure 5-12 64-bit address space map

For example, an address in the range 0 - 7FFFFFFF is below the bar. An address in the range FFFFFFFF - 7FFFFFFF_FFFFFFFF is above the bar. This is basically an alteration to the 2 GB 31-bit terminology that related “below the line” to 24-bit storage, and “above the line” to 31-bit addresses.

The 64-bit address space map differs from the 31-bit address space map in the following ways:

0 - 231** The layout is the same; see Figure 5-11 on page 86.

231 - 2**32** 2 - 4 GB is considered the *bar*. Below the bar can be addressed with a 31-bit address. Above the bar requires a 64-bit address.

232 - 2**35** Reserved area addressable by the Java virtual machine (JVM) using 32-bit pointer compression.

235 - 2**41** The low non-shared area starts at 4 GB and goes to 2**41. A portion of this storage is designed to be used for system storage as an equivalent to LSQA below the 2 GB bar.

Memory objects are allocated in the system area that starts at X’8_00000000’ - 32 GB and ends at X’28_00000000’ - 288 GB using the IARV64 macro with `REQUEST=GETSTOR,LOCALSYSARES=YES`.

241 - 2**50** The Shared Area starts at 2**41 and goes to 2**50 or higher if requested (up to 2**53).

250 - 2**64** The high non-shared area starts at 2**50 or wherever the shared area ends and goes to 2**64.

The area above the bar, as shown in Figure 5-12 on page 87, is designed to keep data and programs. There are several restrictions for programs loaded above the bar. The programs running below the bar can request virtual storage above the bar and access it. To access such an address, the program must be AMODE64.

5.8.1 Region tables

In a 16 EB address space with 64-bit virtual storage addressing, there are three additional levels of translation tables, called *region tables*:

- Region third table (R3T)
- Region second table (R2T)
- Region first table (R1T)

The region tables are 16 KB, and there are 2048 entries per table. Each region has 2 GB.

5.8.2 Page and segment tables

Segment tables and page table formats remain the same as for virtual addresses below the bar. When translating a 64-bit virtual address, after the system has identified the corresponding 2 GB region entry that points to the segment table, the process is the same as that described previously.

5.8.3 User private area

This area above the bar is intended for application data; no programs run above the bar. No system information or system control blocks exist above the bar, either.

The *user private area*, as shown in Figure 5-12 on page 87, includes:

- Low private. The private area below the line.
- Extended private. The private area above the line.
- Low non-shared. The private area just above the bar.
- High non-shared. The private area above the Shared Area.

As users allocate private storage above the bar, it is first allocated from the low non-shared area. Similarly, as the shared area is allocated, it is allocated from the bottom up. This is done to enable applications to have both private and shared memory above the bar, and avoid extra machine cycles to perform dynamic address translation (DAT).

5.9 Segment tables and page tables in 31-bit addressing

Figure 5-13 shows 31-bit virtual address and dynamic address translation. These concepts described in this section.

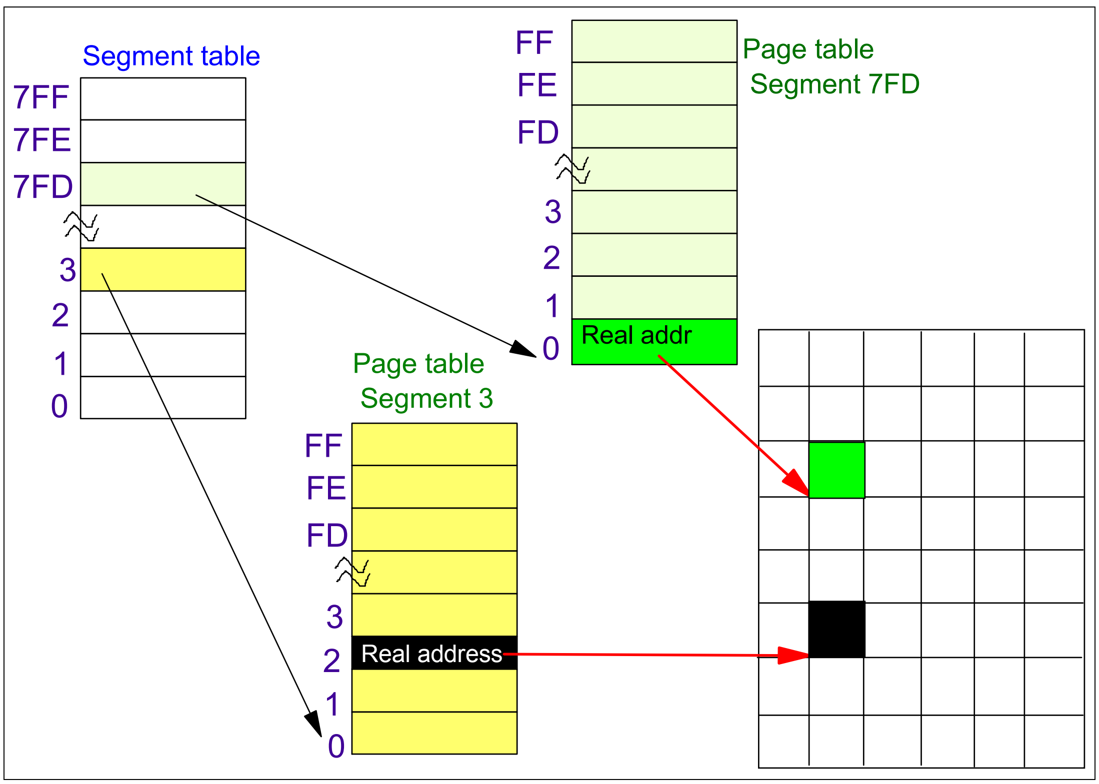

Figure 5-13 31-bit virtual address and dynamic address translation

Main storage is viewed as a long sequence of bits. The sequence 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 31-bit or 64-bit integer values.

An *address space* is a sequence of virtual addresses that is associated with virtual storage. 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 of the addresses used in a program are set up assuming that the program is loaded into CPC memory starting at location 0. In reality, it is not, but this assumption makes decoding the virtual address somewhat easier.

5.9.1 Segment tables and page tables

DAT is the hardware in charge of translating a virtual address during a storage reference into the corresponding real address, using translation tables (segment tables and page tables) prepared by the z/OS component RSM.

Each address space has its own *segment tables* and *page tables*. Each segment table entry has a pointer to the correlated page table. A page table is allocated when the first page on that segment is allocated. There is a maximum of 2048 page tables.

To make this translation easier, the virtual address space is partitioned into segments, each one of 1 MB addresses. Therefore, each 2 GB address space has 2048 segments. Each segment has 256 4 KB pages. Given a virtual address, DAT finds the following information contained in the virtual address:

- Segment index (number of the segment) in the first 11 bits of the address, up to 2047 = b'111 1111 1111'= X'7FF'.
- Page index (number of the page in that segment) in the next 8 bits, up to 255 = b'1111 1111'= X'FF'.
- Byte index (displacement in the page) in the last 12 bits, up to 4095 = b'111111111111' = X'FFF'.

Segment number

The segment number is mapped with an entry into a *segment table* (one entry per segment), with 2048 entries. Each entry is identified in the range 0 - 2047. The entry 0 refers to segment 0, the entry 1 refers to segment 1, and so on.

Each address space has one segment table. Each entry in a segment table points to a *page table* that maps each page of a segment in into an entry table. The system uses a page table with 256 entries.

Because each page is 4 KB, each address space segment has 256 pages. Each entry identifies each page in that segment. Therefore, entry 0 refers to the first page of the segment, entry 1 refers to the second page in the same segment, and so on. The page table entry has the real address of the frame mapping the page, or an invalid bit, when this mapping does not happen. This invalid bit causes a program interrupt known as a *page fault*. When a page fault occurs, the contents of the page are in an auxiliary storage slot.

5.10 Virtual address translation

A *virtual address* identifies a location in virtual storage. When a virtual address is used for access to main storage, it is translated by DAT to a real address in processor memory, as shown in Figure 5-14.

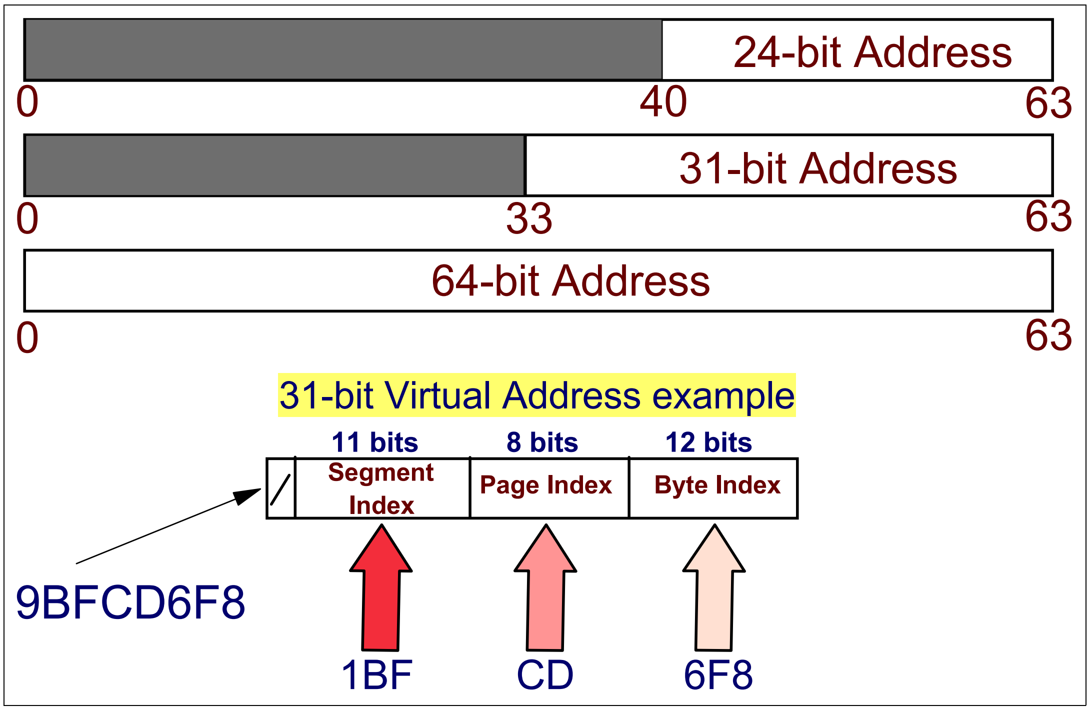

Figure 5-14 Translating 31-bit vIrtual addresses

5.10.1 Address size

An *address size* refers to the maximum number of significant bits that can represent an address. Three sizes of addresses are provided: 24 bit, 31 bit, and 64 bit:

- A 24-bit address can accommodate a maximum of 16,777,216 bytes (16 MB).
- A 31-bit address can address 2,147,483,648 bytes (2 GB).
- A 64-bit address can address 18,446,744,073,709,551,616 bytes (16 EB).

Information: A 24-bit or 31-bit *virtual* address is expanded to 64 bits by appending 40 zeros (24 bit) or 33 zeros (31 bit) on the left before it is translated by DAT process.

5.10.2 31-bit virtual addresses

A virtual address contains the following sections (see Figure 5-14):

- Bits 33 - 43 identify the segment number (segment index).
- Bits 44 - 51 identify the page number in that segment (page index).
- Bits 52 - 63 identify the displacement of the data in that page (byte index).

To translate a 31-bit virtual address (2 GB) into a real address, DAT uses the following bits:

- Bits 33 - 43 as an index in the segment table to find the entry that points to the page table address of that segment

- Bits 44 - 51 as an index in the page table entry that has the frame real address, or that has the invalid bit turned on
- Bits 52 - 63 as the displacement of the data from the beginning of the frame, to be added to the frame real address to get the real address

5.10.3 64-bit virtual address

In a 16 EB address space with 64-bit virtual storage addressing, there are three additional levels of translation tables, called *region tables*. They are known as the region third table (R3T), the region second table (R2T), and the region first table (R1T). The region tables are 16 KB, and there are 2048 entries per table. Figure 5-15 illustrates the table hierarchy and sizes.

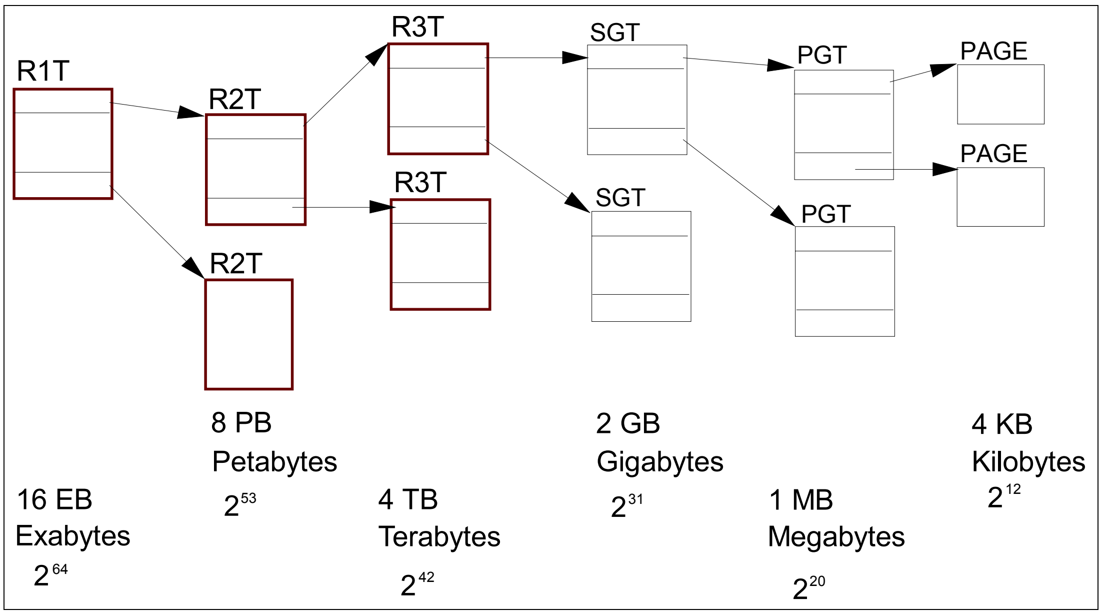

Figure 5-15 64-bit dynamic address translation

Segment table (SGT) and page table (PGT) formats remain the same as for 31-bit virtual address. When translating a 64-bit virtual address, and after you have identified the corresponding 2 GB region entry that points to SGT, the process is the same as described previously.

RSM only creates the additional levels of region tables when it is necessary to back virtual storage that is mapped.

Up to five lookup tables can be needed by DAT to accomplish translation, but the translation only starts from the table that provides translation for the highest usable virtual address in the address space.

5.10.4 Translating a 64-bit virtual address

Up to three additional levels of DAT tables, called *region tables*, are used for translating 64-bit virtual addresses. With 64-bit virtual addressing, there are now three more 11-bit region indexes to the three region tables, as illustrated in Figure 5-16 on page 93.

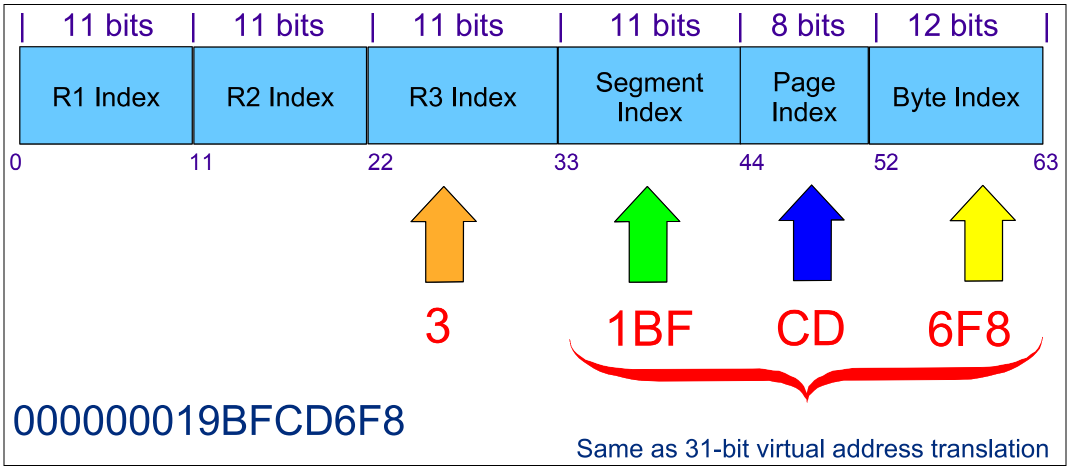

Figure 5-16 Translating a virtual address

To translate a 64-bit virtual address into a real address, DAT uses the following values:

- Bits 0 - 10 are the first region index into the R1T table.
- Bits 11 - 21 are the second region index into the R2T table.
- Bits 22 - 32 are the third region index into the R3T table.
- Bits 33 - 43 are the segment index into the SGT.
- Bits 44 - 51 are the page index into the PGT.
- Bits 52 - 63 indicate the data displacement into the page itself.

5.10.5 1 MB and 2 GB Large Pages

If you have storage-class memory (SCM) in your system, z/OS supports pageable 1 MB pages. With SCM, 1 MB pageable large pages are paged out and back in as 1 MB pageable large pages when requested, or when a storage constraint occurs. If SCM is not available, 1 MB pageable large pages are paged out and back in as 4 KB pageable pages.

2 GB large page-backed memory objects are also supported, but are fixed and not pageable. See *MVS Initialization and Tuning Reference*, SA23-1380.

5.11 Multiprogramming and multiprocessing

The control program creates one *task* in the address space as a result of initiating execution of the job step (the job step task), as shown in Figure 5-17. You can create additional tasks in your program. However, if you do not, the job step task is the only task in the address space being run. The benefits of a multiprogramming environment are still available even with only one task in the job step. Work is still being performed for other address spaces when your task is waiting for an event, such as an input operation, to occur.

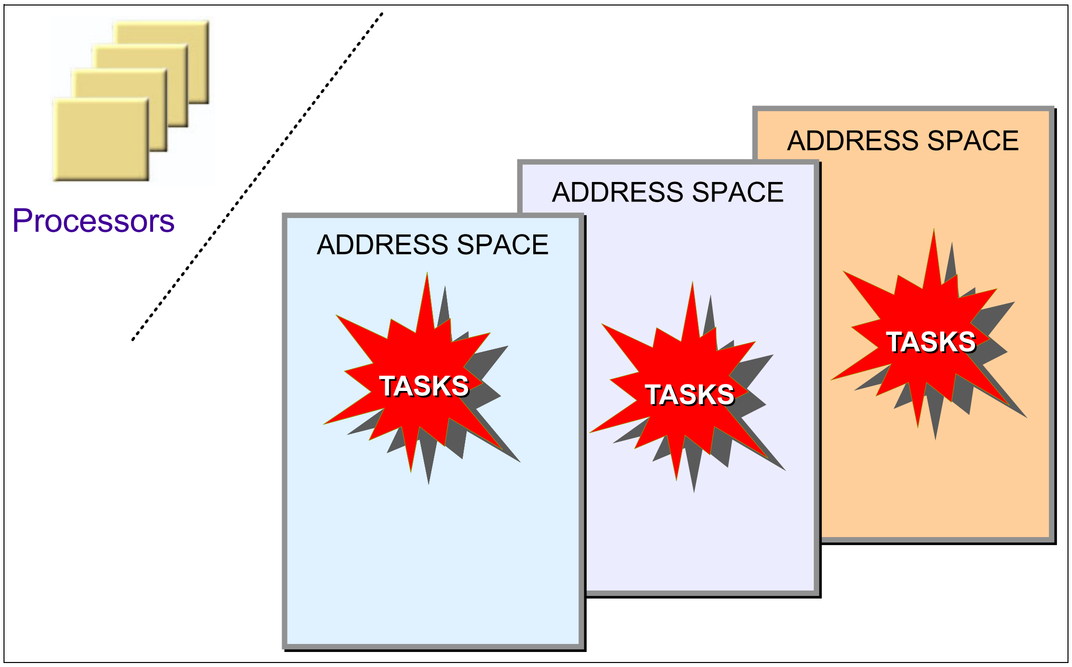

Figure 5-17 Multiprogramming and multiprocessing

The benefit of creating additional tasks in the job step is that more tasks are competing for control. When a wait state occurs in one of your tasks, it is not necessarily a task from another address space that gets control; instead, it can be one of your tasks, a portion of your job.

Only one processor at a time can run the same task. However, several processors running different tasks can run the same program (just as people driving different cars going to different places can share the same street).

5.11.1 Multiprogramming

Multiprogramming means that many tasks can be in a system at the same time, with each task running programs in its own address space (or sometimes in the same address space). In a CPC with only one processor (such as a city with only one car), only one of these tasks can be active at a time.

However, the active task can lose control of the processor at any time (for example, because of I/O requests that place the task in a wait state, meaning it is not a candidate to get the processor). z/OS then selects which task gets control next, based on a number called *dispatching priority*.

5.11.2 Multiprocessing

Multiprocessing is a logical expansion of multiprogramming. Multiprocessing refers to the execution of more than one task simultaneously on more than one processor (which is like a city with several cars). All processors operate under a single copy of z/OS, and share the same memory, as shown in Figure 5-18.

5.12 Program compile, link edit, and execution

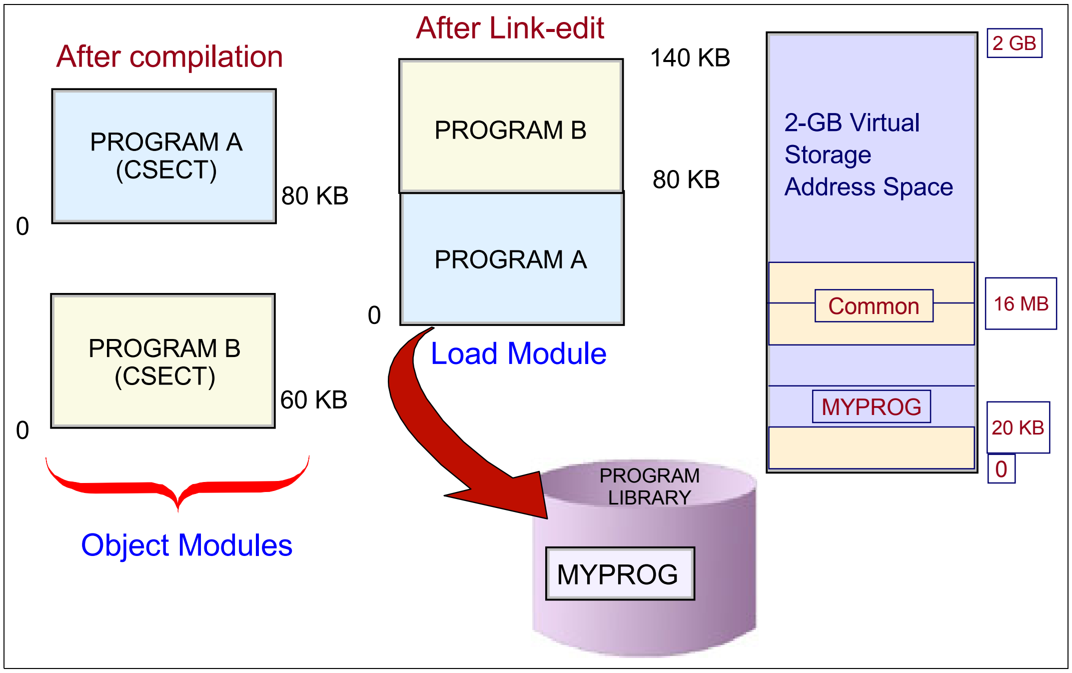

Figure 5-18 Running a program after compiling and link-editing

5.12.1 Program compile, link edit, and execution

A z/OS system can appear to be one big block of code that drives your processor, but z/OS is a complex system composed of many different smaller blocks of code. Each of those smaller blocks of code performs a specific (specialized) function in the system.

Each module of symbolic language code is first assembled or compiled by one of the language translators or the assembler. The input to a language translator is a *source module*. The output from a language translator is an *object module*, made of control sections (CSECTs).

The Binder is a z/OS program that accepts object modules, control statements, and options as input. It combines these object modules, according to the requirements defined by the control statements and options, into a single output load module (executable code). This module is stored in a program library, and loaded into storage for execution by the z/OS component program management loader. A load module can also be an input to the Binder.

Each system function comprises one or more load modules.This is also true for an installation application. In a z/OS environment, a load module represents the basic unit of machine-readable executable code. Load modules are created by combining one or more object modules and processing them with a link-edit utility (Binder).

The link-editing of modules is a process that resolves external references and addresses. Therefore, the functions on your system are one or more object modules that have been combined and link-edited.

5.13 Pervasive Encryption

Pervasive encryption provides a transparent and consumable approach to enable extensive encryption of data in flight and at rest to simplify and reduce the costs associated with protecting data and achieving compliance mandates.

To achieve this standard for encryption, IBM Z delivered several capabilities integrated throughout the z14 stack in the hardware, OS, and middleware, as shown in Figure 5-19.

The on-chip cryptography engine, used for the compression and cryptography feature (CPACF), which offers a set of symmetric cryptographic functions for encrypting and decrypting of clear key. The CPACF has both the cryptographic suite and performance characteristics that can enable bulk encryption of sensitive business data that makes it possible to fortify, intrinsically protecting business data using encryption technology.

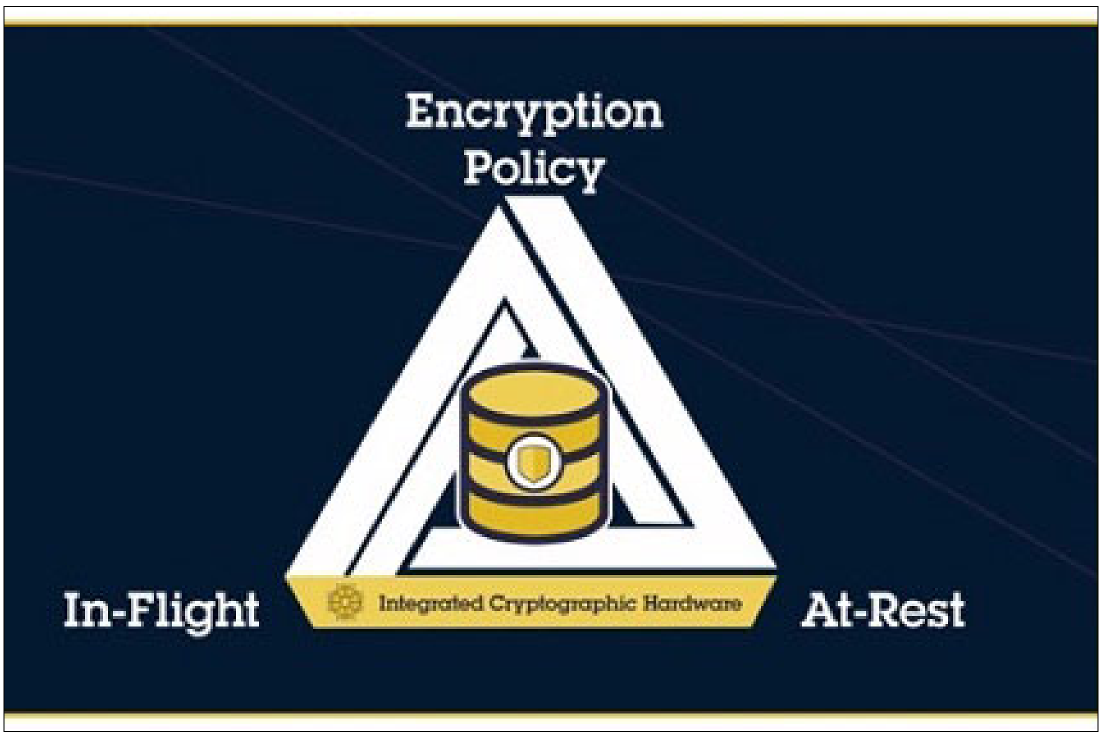

Figure 5-19 Pervasive Encryption

Bulk file and data set encryption were placed at a point in z/OS where the encryption would be transparent to applications and highly optimized for performance.

In addition to helping organizations protect all of their digital assets, pervasive encryption can decouple identification and classification from the process of encryption and reduce the risk of unidentified or misclassified data. It also makes sensitive data within the enterprise more difficult for attackers to identify because it’s all encrypted.

5.13.1 z/OS Encryption Capabilities

z/OS is designed to provide policy-based encryption options that take full advantage of the encryption capabilities in the z14 platform, and can help clients protect their critical business data. The following new capabilities are included:

- Enhanced data protection for many z/OS data sets, zFS file systems, and Coupling Facility structures gives users the ability to encrypt data without needing to change applications to embed encryption APIs within applications.
- z/OS policy controls make it possible to use pervasive encryption to help protect user data and simplify the task of compliance.
- z/OS Communications Server includes encryption-readiness technology to enable z/OS administrators to determine which TCP and Enterprise Extender traffic patterns to and from their z/OS systems meet approved encryption criteria and help simplify the task of compliance.

Some of the following design advantages are built into z/OS data set encryption:

- Uses CPACF and protected key, which means that key material is not visible in clear text format, offering a higher level of protection and the high throughput of encryption using CPACF
- Is designed to protect data in a way that is aligned with customers' current access control mechanisms offering a more straightforward configuration experiences
- Is designed to perform efficiently at speed
- Can enable encryption without requiring application or database changes
- Allows data to remain encrypted throughout its journey. For instance, with z/OS data set encryption, any data replicated, or backed up or migrated, remains encrypted
- In-memory buffer content is not encrypted, so every data access does not require an encrypt or a decrypt operation; this design helps reduce the overall cost of encryption
- Can be configured such that encryption keys are owned and managed by logical organizational environment (for example, production versus test) providing cryptographic separation from other environments
- Can help simplify clients' compliance efforts

Data set encryption enables encryption of files in bulk through the access method. z/OS data set encryption is designed to offer high throughput, low-cost encryption. It is intended to be more accessible to the organization than many other forms of encryption. It is designed to be transparent to the application, requiring no changes to application code. z/OS data set encryption enables customers to encrypt data at course scale without the need to perform data identification and classification first.

z/OS data set encryption is designed to use AES 256.

5.13.2 z/OS DFSMS and pervasive encryption

z/OS DFSMS introduces pervasive encryption of data at rest for extended format data sets accessed through access methods without requiring application changes. z/OS data set encryption through RACF command and SMS policies can be used to identify the data sets or groups of data sets that require encryption.

You can protect viewing the data in the clear based on authorization access to the key label that is associated with the data set used by the access methods to encrypt and decrypt the data. The data set owner specifies an encryption key label, which refers to an AES 256-bit encryption key that exists in the ICSF key repository.

z/OS DFSMS data set encryption can be used to encrypt the following types of data sets:

- Sequential extended format data sets that are accessed through BSAM and QSAM
- VSAM extended format data sets (KSDS, ESDS, RRDS, VRRDS, LDS) that are accessed through base VSAM and VSAM/RLS

To create an encrypted data set, you must assign a key label to the data set when it is created (first allocated). A key label can be specified through any of the following methods:

- RACF data set profile
- JCL, dynamic allocation, TSO ALLOCATE, IDCAMS DEFINE
- SMS data class

Encrypted data sets must be SMS-managed extended format; they can be in compressed format, also.

DFSMSdss and DFSMShsm support backup and migration of encrypted data sets while preserving the data in encrypted form.

Encrypted data sets must be SMS-managed extended format.

5.13.3 DB2 and IMS and z/OS data set encryption

DB2 is designed to transparently encrypt data at rest without database downtime or requiring the administrator to redefine objects, which could cause disruption to operations. This includes the ability to transparently encrypt its logs, catalog, directory, tables, and indexes, including all data types such as large binary objects transparently. In addition, for maximum availability, rekeying of data keys can be performed non-disruptively without taking DB2 databases offline.

IMS V14 supports z/OS data set encryption for select data sets. The IMS 15 Quality Partnership Program (QPP) offering also supports these capabilities. Also, z/OS data set encryption allows customers to take advantage of transparent encryption of select IMS data sets.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 100. Note that some of the documents referenced here might be available in softcopy only.

- *ABCs of z/OS System Programming Volume 2*, SG24-6982
- *ABCs of z/OS System Programming Volume 3*, SG24-6983
- *ABCs of z/OS System Programming: Volume 4*, SG24-6984
- *ABCs of z/OS System Programming: Volume 5*, SG24-6985
- *ABCs of z/OS System Programming Volume 6*, SG24-6986
- *ABCs of z/OS System Programming Volume 7*, SG24-6987
- *ABCs of z/OS System Programming Volume 8*, SG24-6988
- *ABCs of z/OS System Programming: Volume 9*, SG24-6989
- *ABCs of z/OS System Programming Volume 10*, SG24-6990
- *ABCs of z/OS System Programming Volume 11*, SG24-6327
- *ABCs of z/OS System Programming Volume 12*, SG24-7621
- *ABCs of z/OS System Programming Volume 13*, SG24-7717

Other publications

These publications are also relevant as further information sources:

- *z/OS Planning for Installation*, GA32-0890
- *z/OS MVS Initialization and Tuning Reference*, SA23-1380
- *z/OS MVS JCL Reference*, SA23-1385
- *z/OS MVS JCL User’s Guide*, SA23-1386
- *z/OS Security Server RACF System Programmer’s Guide*, SA23-2287
- *z/OS TSO/E Command Reference*, SA32-0975
- *z/OS TSO/E Customization*, SA32-0976
- *z/OS TSO/E Primer*, SA32-0984
- *z/OS TSO/E REXX User's Guide*, SA32-0982
- *z/OS TSO/E User's Guide*, SA32-0971
- *z/OS Using REXX and z/OS UNIX System Services*, SA23-2283
- *IBM ServerPac Using the Installation Dialog*, SA23-2278

- *OSA-Express Customer's Guide and Reference*, SA22-7935
- *Interactive System Productivity Facility Getting Started*, SC34-4440
- *z/OS ISPF Planning and Customizing*, GC19-3623
- *z/OS ISPF Dialog Developer's Guide*, SC19-3619
- *z/OS ISPF SCLM Project Manager's and Developer's Guide*, SC19-3625
- *z/OS Distributed File Services zFS Administration*, SC23-6887
- *z/OS DFSMS Managing Catalogs*, SC23-6853
- *z/OS ISPF Edit and Edit Macros*, SC19-3621
- *z/OS ISPF User's Guide Volume I*, SC19-3627
- *z/OS ISPF User's Guide Volume II*, SC19-3628

Online resources

These websites are also relevant as further information sources:

- The ShopzSeries web address:

  `https://www14.software.ibm.com/webapp/ShopzSeries/ShopzSeries.jsp`
- The PMA website:

  `http://www-03.ibm.com/systems/z/os/zos/features/smpe/pma/index.html`

How to get IBM Redbooks

You can search for, view, or download IBM Redbooks, IBM Redpapers, Technotes, draft publications, and Additional materials, and order hardcopy Redbooks, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads:

ibm.com/support

IBM Global Services:

ibm.com/services

Redbooks

ABCs of z/OS System Programming Volume 1

(0.2"spine)
0.17"<->0.473"
90<->249 pages

SG24-6981-04

ISBN 0738442763

Printed in U.S.A.