Contents

Preface

This course provides students of information systems technology with the background knowledge and skills necessary to begin using the basic facilities of a mainframe computer.

For optimal learning, students are assumed to have successfully completed an introductory course in computer system concepts, such as computer organization and architecture, operating systems, data management, or data communications. They should also have successfully completed courses in one or more programming languages, and be PC literate.

Note that this course can also be used as a prerequisite for courses in advanced topics such as compiler algorithms, or for internships and special studies.

Others who will benefit from this course include data processing professionals who have experience with non-mainframe platforms, or who are familiar with some aspects of the mainframe but want to become knowledgeable with other facilities and benefits of the mainframe environment.

When moving through this course, instructors are encouraged to alternate between course, lecture, discussions, and hands-on exercises. The instructor-led discussions and hands-on exercises are an integral part of the learning experience, and can include topics not covered in this course.

After completing this course, students will have received:

- A general introduction to mainframe concepts, usage, and architecture
- A comprehensive overview of z/OS, a widely used mainframe operating system
- An understanding of mainframe workloads and an overview of the major middleware applications in use on mainframes today
- The basis for subsequent course work in more advanced, specialized areas of z/OS, such as system administration or application programming

How this course is organized

This course is organized in four parts, as follows:

- **Part 1. "Introduction to z/OS and the mainframe environment"** provides an overview of the types of workloads commonly processed on the mainframe, such as batch jobs and online transactions. This part of the course helps students explore the

user interfaces of z/OS, a widely used mainframe operating system. Discussion topics include TSO/E and ISPF, UNIX interfaces, job control language, file structures, and job entry subsystems. Special attention is paid to the users of mainframes and to the evolving role of mainframes in today's business world.

- **Part 2. "Application programming on z/OS"** introduces the tools and utilities for developing a simple program to run on z/OS. This part of the course guides the student through the process of application design, choosing a programming language, and using a runtime environment.
- **Part 3. "Online workloads for z/OS"** examines the major categories of interactive workloads processed by z/OS, such as transaction processing, database management, and Web-serving. This part of the course includes discussions of several popular middleware products, including DB2®, CICS®, and WebSphere® Application Server.
- **Part 4. "System programming on z/OS"** provides topics to help the student become familiar with the role of the z/OS system programmer. This part of the course includes discussions of system libraries, starting and stopping the system, security, network communications and the clustering of multiple systems. Also provided is an overview of mainframe hardware systems, including processors and I/O devices.

In this course, we use simplified examples and focus mainly on basic system functions. Hands-on exercises are provided throughout the course to help students explore the mainframe style of computing. Exercises include entering work into the system, checking its status, and examining the output of submitted jobs.

How each topic is organized

Each topic follows a common format:

- Objectives for the student
- Topics that teach a central theme related to mainframe computing
- Summary of the main ideas of the topic
- A list of key terms introduced in the topic
- Questions for review to help students verify their understanding of the material
- Topics for further discussion to encourage students to explore issues that extend beyond the topic objectives
- Hands-on exercises intended to help students reinforce their understanding of the material

System programming on z/OS

In this part we reveal the inner workings of z/OS with discussions of system libraries, change management, and procedures for starting (IPLing) and stopping a z/OS system. This part also includes chapters on hardware details and virtualization, and the clustering of multiple z/OS systems in a *sysplex*.

1

Overview of system programming

Objective: As a z/OS system programmer, you need to know how to start the system and how to bring it down again. You must also understand the role of z/OS system libraries, such as PARMLIB and linklist, and how these libraries affect the processing of z/OS. Also important is a familiarity with SMP/E, the z/OS utility that helps you coordinate changes to the system software.

After completing this topic, you will be able to:

- List the major responsibilities of a z/OS system programmer
- Discuss the system libraries, their use, and methods for managing their content
- Explain how proper installation and maintenance are the basis for high availability in the z/OS environment.
- Describe the process of IPLing a system.

1.1 The role of the system programmer

The system programmer is responsible for managing the mainframe hardware configuration, and installing, customizing, and maintaining the mainframe operating system. Installations need to ensure that their system and its services are available and operating to meet service level agreements. Installations with 24-hour, 7-day operations need to plan for minimal disruption of their operation activities.

In this topic, we examine several areas of interest for the would-be z/OS system programmer. While this course cannot cover every aspect of system programming, it's important to learn that the job of the z/OS system programmer is very complex and requires skills in many aspects of the system, such as:

- Device I/O configurations
- Processor configurations
- Console definitions
- System libraries where the software is placed
- System data sets and their placement
- Customization parameters that are used to define your z/OS configuration
- Installation and maintenance using SMP/E.
- Security administration

As shown in Figure 1-1 on page 9, the role of system programmer usually includes some degree of involvement in all of the following aspects of system operation:

- "Customizing the system" on page 10
- "Managing system performance" on page 21
- "Configuring I/O devices" on page 22
- "Following a process of change control" on page 23
- "Configuring consoles" on page 26
- "Initializing the system" on page 31

We discuss mainframe hardware configurations in Chapter 3, "Hardware systems and LPARs" on page 77.

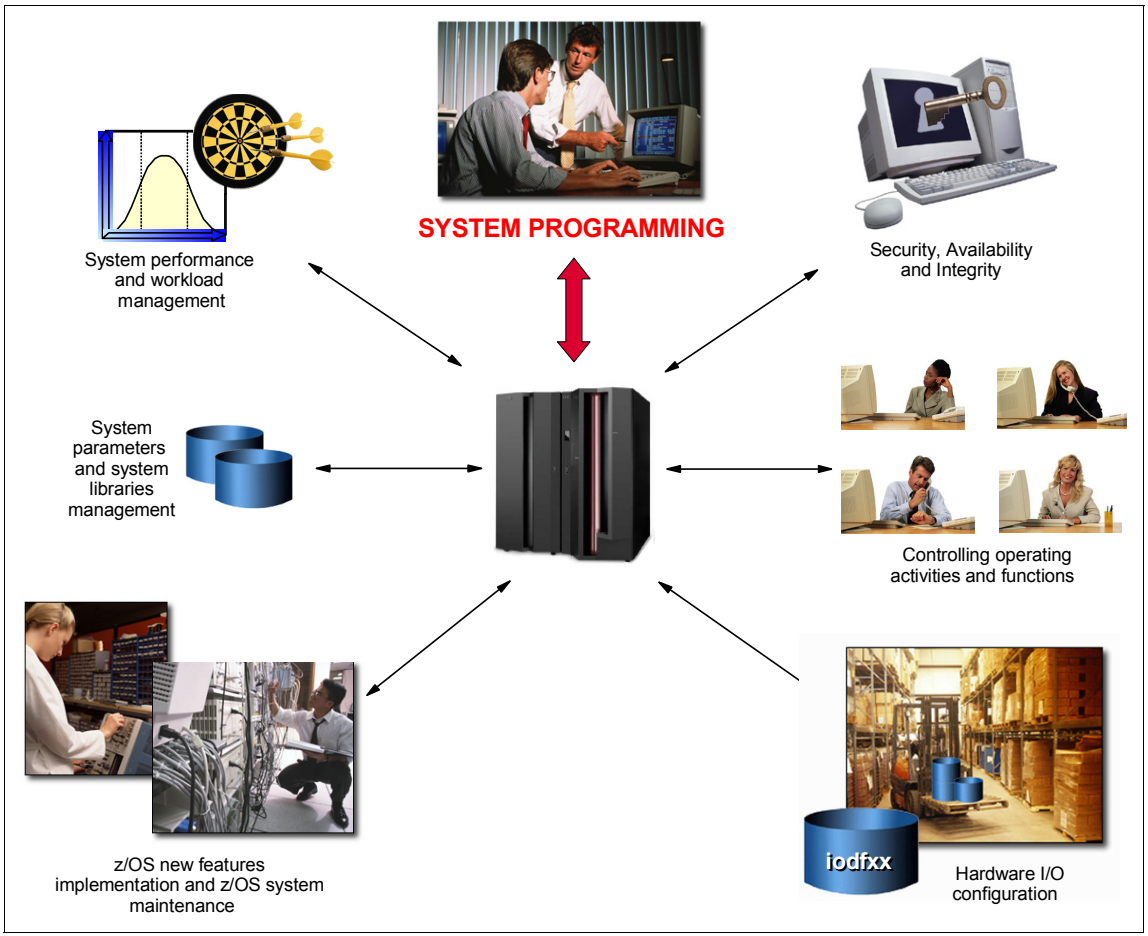


Figure 1-1 Some areas in which the system programmer is involved

1.2 What is meant by separation of duties

In a large z/OS installation, there is usually a “separation of duties” both among members of the system programming staff, and between the system programming department and other departments in the IT organization.

A typical z/OS installation includes the following roles and more:

- z/OS system programmer
- CICS system programmer
- Database system programmer
- Database administrator

- Network system programmer
- Automation specialist
- Security manager
- Hardware management
- Production control analyst
- System operator
- Network operator
- Security administrator
- Service manager

In part, the separation is an audit requirement—ensuring that one person does not have too much power on a system.

When a new application is to be added to a system, for example, a number of tasks need to be performed before the application can be used by end users. A production control analyst is needed to add batch applications into the batch scheduling package, add the new procedures to a procedure library, and set up the operational procedures. The system programmer is needed to perform tasks concerned with the system itself, such as setting up security privileges and adding programs to system libraries. The programmer is also involved with setting up any automation for the new application.

On a test system, however, a single person might have to perform all the roles, including being the operator, and this is often the best way to learn how everything works.

1.3 Customizing the system

This section describes the following topics:

- System libraries where the software is located
- System data sets and their placement
- I/O device configuration
- Console definitions
- Customization parameters used to define the z/OS configuration
- z/OS implementation and maintenance

As can be seen in Figure 1-2 on page 11, different types of data exist in a system. First there is the z/OS software as supplied by IBM. This is usually installed to a series of disk volumes known as the system residence volumes (SYSRES).

Much of the flexibility of z/OS is built on these SYSRES sets. They make it possible to apply maintenance to a new set that is cloned from the production set while the current set is running production work. A short outage can then be taken to IPL from the new set—and the maintenance has been implemented! Also, the change can be backed out by IPLing from the old set.

Fixes to z/OS are managed with a product called System Management Program/Extended (SMP/E). Indirect cataloging using system symbols is used so that a particular library is cataloged as being on, for example, SYSRES volume 2, and the name of that volume is resolved by the system at IPL time from the system symbols. Symbols are discussed in 1.3.11, “What system symbols are” on page 19.

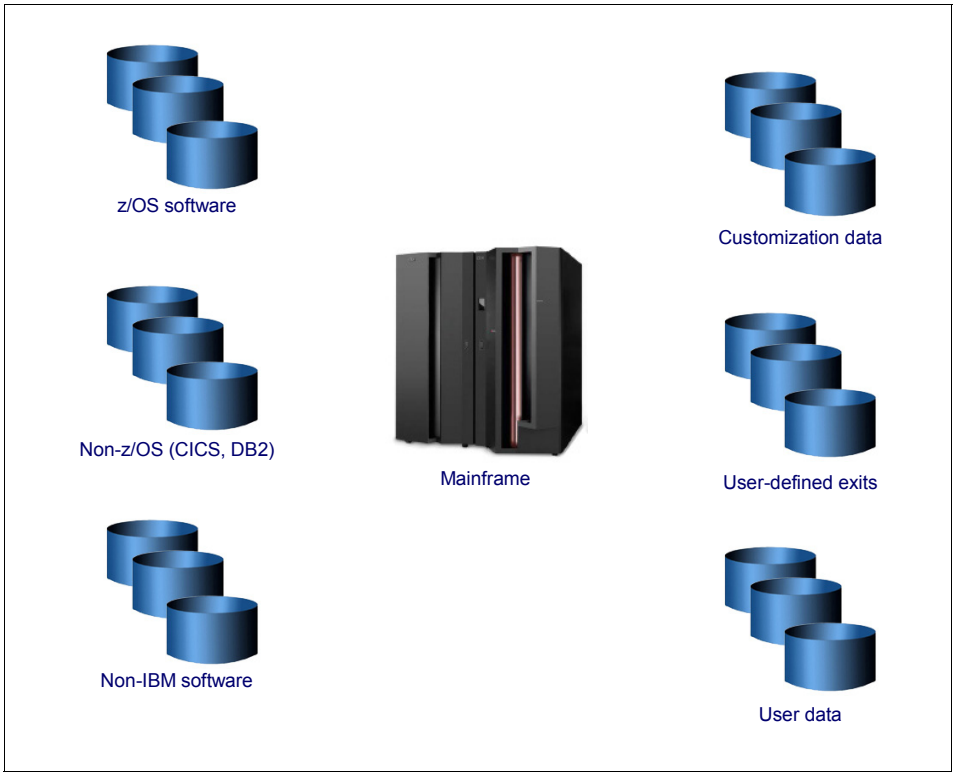


Figure 1-2 Types of data

Another group of volumes are the non-z/OS and non-IBM software volumes. These may be combined into one group. The majority of non-z/OS software is not usually on the SYSRES volumes, as the SYSRES sets are usually managed as one entity by SMP/E. The other software is usually managed separately. These volumes do not form part of the SYSRES sets, and therefore there is only one copy of each library. As many volumes as required can be added to this group, each with an individual disk name.

Customization data refers to *system libraries* such as SYS1.PARMLIB, SYS1.PROCLIB, plus the master catalog, the IODF, page data sets, JES spools, and other items essential to the running of the system. It is also where SMP/E data is stored to manage the software. These data sets are not always located on separate DASD volumes from IBM-supplied z/OS software; some installations place the PARMLIB and

PROCLIB on the first SYSRES pack, others place them on the master catalog pack or elsewhere. This is a matter of choice and is dependent on how the SYSRES volumes are managed. Each installation will have a preferred method. On many systems, some of the IBM-supplied defaults are not appropriate so they need to be modified. User exits and user modifications (usermods) are made to IBM code so that it will behave as the installation requires. The modifications are usually managed using SMP/E.

Finally, there is *user data*, which is usually the largest pool of disk volumes. This is not part of the system libraries, but is presented here for completeness. It contains production, test and user data. It is often split into pools and managed by System Managed Storage (SMS), which can target data to appropriately managed volumes. For example, production data can be placed on volumes which are backed up daily, whereas user data may only be captured weekly and may be migrated to tape after a short period of inactivity to free up the disk volumes for further data.

1.3.1 z/OS system libraries

z/OS has many standard system libraries, such as: SYS1.PARMLIB, SYS1.LINKLIB, SYS1.LPALIB, SYS1.PROCLIB, and SYS1.NUCLEUS. Some of these are related to IPL processing, while others are related to the search order of invoked programs or to system security, as described here:

- SYS1.PARMLIB contains control parameters for the whole system.
- SYS1.LINKLIB has many execution modules of the system.
- SYS1.LPALIB contains the system execution modules that are loaded into the link pack area when the system initializes.
- SYS1.PROCLIB contains JCL procedures distributed with z/OS.
- SYS1.NUCLEUS has the basic supervisor modules of the system.

1.3.2 SYS1.PARMLIB

SYS1.PARMLIB is a required partitioned data set that contains IBM-supplied and installation-created members. It must reside on a direct access volume, which can be the system residence volume. PARMLIB is an important data set in a z/OS operating system, and can be thought of as performing a function similar to /etc on a UNIX system.

The purpose of the PARMLIB is to provide many initialization parameters in a pre-specified form in a single data set, and thus minimize the need for the operator to enter parameters.

All parameters and members of the SYS1.PARMLIB data set are described in *z/OS MVS Initialization and Tuning Reference,* SA22-7592. Some of the most important PARMLIB members are discussed in this section.

1.3.3 Link pack area (LPA)

The link pack area (LPA) is a section of the common area of an address space. It exists below the system queue area (SQA) and consists of the pageable link pack area (PLPA), then the fixed link pack area (FLPA), if one exists, and finally the modified link pack area (MLPA).

Link pack area (LPA) modules are loaded in common storage, shared by all address spaces in the system. Because these modules are reentrant and are not self-modifying, each can be used by a number of tasks in any number of address spaces at the same time. Modules found in LPA do not need to be brought into virtual storage because they are already in virtual storage.

Modules placed anywhere in the LPA are always in virtual storage, and modules placed in FLPA are also always in central storage. LPA modules must be referenced very often in order to prevent their pages from being stolen. When a page in LPA (other than in FLPA) is not continually referenced by multiple address spaces, it tends to be stolen.

1.3.4 Pageable link pack area (PLPA)

The PLPA is an area of common storage that is loaded at IPL time (when a cold start is done and the CLPA option is specified). This area contains read-only system programs, along with any read-only reenterable user programs selected by an installation that can be shared among users of the system. The PLPA and extended PLPA contain all members of SYS1.LPALIB and other libraries that are specified in the active LPALSTxx through the LPA parameter in IEASYSxx or from the operator's console at system initialization (this would override the PARMLIB specification).

You may use one or more LPALSTxx members in SYS1.PARMLIB to concatenate your installation's program library data sets to SYS1.LPALIB. You can also use the LPALSTxx member to add your installation's read-only reenterable user programs to the pageable link pack area (PLPA). The system uses this concatenation, which is referred to as the *LPALST concatenation*, to build the PLPA during the nucleus initializing process. SYS1.LPALIB must reside in a direct access volume, which can be the system residence volume.

Figure 1-3 shows an example of the LPALSTxx member.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT     SYS1.PARMLIB(LPALST7B) - 01.03            Columns 00001 00072
Command ===>                                    Scroll ===> CSR
****** ***************************** Top of Data ******************************
000200 SYS1.LPALIB,
000220 SYS1.SERBLPA,
000300 ISF.SISFLPA,
000500 ING.SINGMOD3,
000600 NETVIEW.SCNMLPA1,
000700 SDF2.V1R4M0.SDGILPA,
000800 REXX.SEAGLPA,
001000 SYS1.SIATLPA,
001100 EOY.SEOYLPA,
001200 SYS1.SBDTLPA,
001300 CEE.SCEELPA,
001400 ISP.SISPLPA,
001600 SYS1.SORTLPA,
001700 SYS1.SICELPA,
001800 EUV.SEUVLPA,
001900 TCPIP.SEZALPA,
002000 EQAW.SEQALPA,
002001 IDI.SIDIALPA,
002002 IDI.SIDILPA1,
002003 DWW.SDWWLPA(SBOX20),
002010 SYS1.SDWWDLPA,
002020 DVG.NFTP230.SDVGLPA,
002200 CICSTS22.CICS.SDFHLPA(SBOXD3)
****** ***************************** Bottom of Data ***************************
```

Figure 1-3 Example of LPALST PARMLIB member

1.3.5 Fixed link pack area (FLPA)

The FLPA is loaded at IPL time, with those modules listed in the active IEAFIXxx member of SYS1.PARMLIB. This area should be used only for modules that significantly increase performance when they are fixed rather than pageable. The best candidates for the FLPA are modules that are infrequently used, but are needed for fast response.

Modules from the LPALST concatenation, the linklist concatenation, SYS1.MIGLIB, and SYS1.SVCLIB can be included in the FLPA. FLPA is selected through specification of the FIX parameter in IEASYSxx, which is appended to IEAFIX to form the IEAFIXxx PARMLIB member, or from the operator's console at system initialization.

Figure 1-4 shows an example of an IEAFIX member; some of the modules for FLPA belong to the SYS1.LPALIB library.

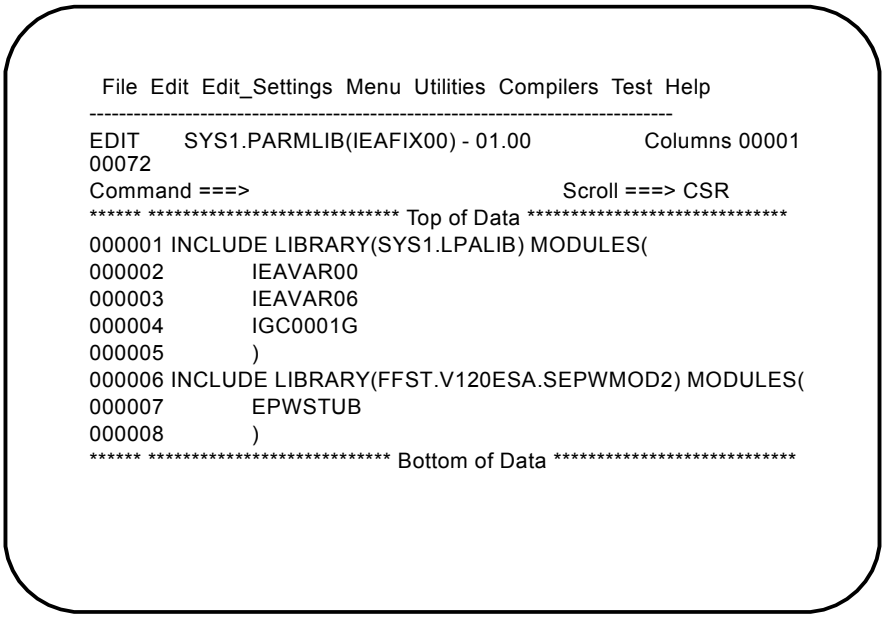


Figure 1-4 Example of an IEAFIX PARMLIB member

1.3.6 Modified link pack area (MLPA)

The MLPA can be used to contain reenterable routines from APF-authorized libraries that are to be part of the pageable extension to the link pack area during the current IPL. Note that the MLPA exists only for the duration of an IPL. Therefore, if an MLPA is desired, the modules in the MLPA must be specified for each IPL (including quick start and warm start IPLs). When the system searches for a routine, the MLPA is searched before the PLPA. The MLPA can be used at IPL time to temporarily modify or update the PLPA with new or replacement modules.

1.3.7 SYS1.PROCLIB

SYS1.PROCLIB is a required partitioned data set that contains the JCL procedures used to perform certain system functions. The JCL can be for system tasks or processing program tasks invoked by the operator or the programmer.

1.3.8 The master scheduler subsystem

The *master scheduler subsystem* is used to establish communication between the operating system and the primary job entry subsystem, which can be JES2 or JES3.

When you start z/OS, master initialization routines initialize system services, such as the system log and communication task, and start the master scheduler address space, which becomes address space number one (ASID=1).

Then, the master scheduler may start the job entry subsystem (JES2 or JES3). JES is the primary job entry subsystem. On many production systems JES is not started immediately; instead, the automation package starts all tasks in a controlled sequence. Then other defined subsystems are started. All subsystems are defined in the IEFSSNxx member of PARMLIB. These subsystems are *secondary subsystems*.

An initial MSTJCL00 load module can be found in SYS1.LINKLIB library. If modifications are required, the recommended procedure is to create a MSTJCLxx member in PARMLIB. The suffix is specified by the MSTRJCL parameter in the IEASYSxx member of PARMLIB. The MSTJCLxx member is commonly called *master JCL*. It contains data definition (DD) statements for all system input and output data sets that are needed to do the communication between the operating system and JES.

Example 1-1 shows an example of an MSTJCLxx member.

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT       SYS1.PARMLIB(MSTJCL00) - 01.07               Columns 00001 00072
Command ===>                                              Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //MSTRJCL  JOB MSGLEVEL=(1,1),TIME=1440
000200 //         EXEC PGM=IEEMB860,DPRTY=(15,15)
000300 //STCINRDR DD SYSOUT=(A,INTRDR)
000400 //TSOINRDR DD SYSOUT=(A,INTRDR)
000500 //IEFPDSI  DD DSN=SYS1.PROCLIB,DISP=SHR
000600 //         DD DSN=CPAC.PROCLIB,DISP=SHR
000700 //         DD DSN=SYS1.IBM.PROCLIB,DISP=SHR
000800 //IEFJOBS  DD DSN=SYS1.STCJOBS,DISP=SHR
000900 //SYSUADS  DD DSN=SYS1.UADS,DISP=SHR
****** **************************** Bottom of Data ****************************
```

Example 1-1 Sample master JCL

When the master scheduler has to process the start of a started task, the system determines whether the START command refers to a procedure or to a job. If the IEFJOBS DD exists in the MSTJCLxx member, the system searches the IEFJOBS DD concatenation for the member requested in the START command.

If there is no member by that name in the IEFJOBS concatenation, or if the IEFJOBS concatenation does not exist, the system searches the IEFPDSI DD for the member requested in the START command. If a member is found, the system examines the first record for a valid JOB statement and, if one exists, uses the member as the JCL source for the started task. If the member does not have a valid JOB statement in its first record,

the system assumes that the source JCL is a procedure and creates JCL to invoke the procedure.

After the JCL source has been created (or found), the system processes the JCL. As shipped, MSTJCL00 contains an IEFPDSI DD statement that defines the data set that contains procedure source JCL for started tasks. Normally this data set is SYS1.PROCLIB; it may be a concatenation. For useful work to be performed, SYS1.PROCLIB must at least contain the procedure for the primary JES, as shown in next section.

1.3.9 A job procedure library

SYS1.PROCLIB contains the JES2 cataloged procedure. This procedure defines the job-related procedure libraries, as shown in Example 1-2.

Example 1-2 How to specify procedure libraries in the JES2 procedure

```
//PROC00 DD  DSN=SYS1.PROCLIB,DISP=SHR
//       DD  DSN=SYS3.PROD.PROCLIB,DISP=SHR
//PROC01 DD  DSN=SYS1.PROC2,DISP=SHR
...
//PROC99 DD  DSN=SYS1.LASTPROC,DISP=SHR
...
```

Many installations have very long lists of procedure libraries in the JES procedure. This is because JCLLIB is a relatively recent innovation.

Care should be taken as to the number of users who can delete these libraries because JES will not start if one is missing. Normally a library that is in use cannot be deleted, but JES does not hold these libraries although it uses them all the time.

You can override the default specification by specifying this statement:

```
/*JOBPARM PROCLIB=
```

After the name of the procedure library, you code the name of the DD statement in the JES2 procedure that points to the library to be used. For example, in Figure 1-2, let's assume that you run a job in class A and that class has a default proclib specification on PROC00. If you want to use a procedure that resides in SYS1.LASTPROC, you'll need to include this statement in the JCL:

```
/*JOBPARM PROCLIB=PROC99
```

Another way to specify a procedure library is to use the JCLLIB JCL statement. This statement allows you to code and use procedures without using system procedure libraries. The system searches the libraries in the order in which you specify them on the JCLLIB statement, prior to searching any unspecified default system procedure libraries.

Example 1-3 shows the use of the JCLLIB statement.

Example 1-3 Sample JCLLIB statement

```
//MYJOB   JOB
//MYLIBS  JCLLIB    ORDER=(MY.PROCLIB.JCL,SECOND.PROCLIB.JCL)
//S1      EXEC PROC=MYPROC1
...
```

Assuming that the system default procedure library includes SYS1.PROCLIB only, the system searches the libraries for procedure `MYPROC1` in the following order:

1. MY.PROCLIB.JCL
2. SECOND.PROCLIB.JCL
3. SYS1.PROCLIB

1.3.10 Search order for programs

When a program is requested through a system service (like LINK, LOAD, XCTL, or ATTACH) using default options, the system searches for it in the following sequence:

1. **Job pack area (JPA)**. A program in JPA has already been loaded in the requesting address space. If the copy in JPA can be used, it will be used. Otherwise, the system either searches for a new copy or defers the request until the copy in JPA becomes available. (For example, the system defers a request until a previous caller is finished before reusing a serially-reusable module that is already in JPA.)
2. **TASKLIB.** A program can allocate one or more data sets to a TASKLIB concatenation. Modules loaded by unauthorized tasks that are found in TASKLIB must be brought into private area virtual storage before they can run. Modules that have previously been loaded in common area virtual storage (LPA modules or those loaded by an authorized program into CSA) must be loaded into common area virtual storage before they can run.
3. **STEPLIB or JOBLIB.** These are specific DD names that can be used to allocate data sets to be searched ahead of the default system search order for programs. Data sets can be allocated to both the STEPLIB and JOBLIB concatenations in JCL or by a program using dynamic allocation. However, only one or the other will be searched for modules. If both STEPLIB and JOBLIB are allocated for a particular jobstep, the system searches STEPLIB and ignores JOBLIB.

   Any data sets concatenated to STEPLIB or JOBLIB will be searched after any TASKLIB but before LPA. Modules found in STEPLIB or JOBLIB must be brought into private area virtual storage before they can run. Modules that have previously been loaded in common area virtual storage (LPA modules or those loaded by an authorized program into CSA) must be loaded into common area virtual storage before they can run.
4. **LPA**, which is searched in this order:

a. Dynamic LPA modules, as specified in PROGxx members
b. Fixed LPA (FLPA) modules, as specified in IEAFIXxx members
c. Modified LPA (MLPA) modules, as specified in IEALPAxx members
d. Pageable LPA (PLPA) modules, loaded from libraries specified in LPALSTxx or PROGxx

LPA modules are loaded in common storage, shared by all address spaces in the system. Because these modules are reentrant and are not self-modifying, each can be used by any number of tasks in any number of address spaces at the same time. Modules found in LPA do not need to be brought into virtual storage, because they are already in virtual storage.

5. **Libraries in the linklist, as specified in PROGxx and LNKLSTxx.** By default, the linklist begins with SYS1.LINKLIB, SYS1.MIGLIB, and SYS1.CSSLIB. However, you can change this order using SYSLIB in PROGxx and add other libraries to the linklist concatenation. The system must bring modules found in the linklist into private area virtual storage before the programs can run.

The default search order can be changed by specifying certain options on the macros used to call programs. The parameters that affect the search order the system will use are EP, EPLOC, DE, DCB, and TASKLIB.

Related Reading: For more information about these parameters, see the topic on load module search in the IBM publication *z/OS MVS Programming: Assembler Services Guide*. Some IBM subsystems (notably CICS and IMS) and applications (such as ISPF) use these facilities to establish search orders for other programs.

1.3.11 What system symbols are

System symbols are elements that allow different z/OS systems to share PARMLIB definitions while retaining unique values in those definitions. System symbols act like variables in a program; they can take on different values, based on the input to the program. When you specify a system symbol in a shared PARMLIB definition, the system symbol acts as a “placeholder”. Each system that shares the definition replaces the system symbol with a unique value during initialization.

Each system symbol has a name (which begins with an ampersand (&) and optionally ends with a period (.)) and a substitution text, which is the character string that the system substitutes for a symbol each time it appears.

There are two types of system symbols:

dynamic The substitution text can change at any point in an IPL.

static The substitution text is defined at system initialization and remains fixed for the life of an IPL.

There are symbols that are reserved for system use. You can display the symbols in your system by entering the **D SYMBOLS** command. Example 1-4 shows the result of entering this command.

Example 1-4 Partial output of the D SYMBOLS command (some lines removed)

```
HQX7708 -----------------  SDSF PRIMARY OPTION MENU  --
COMMAND INPUT ===> -D SYMBOLS
  IEA007I STATIC SYSTEM SYMBOL VALUES
          &SYSALVL.  = "2"
          &SYSCLONE. = "70"
          &SYSNAME.  = "SC70"
          &SYSPLEX.  = "SANDBOX"
          &SYSR1.    = "Z17RC1"
          &ALLCLST1. = "CANCEL"
          &CMDLIST1. = "70,00"
          &COMMDSN1. = "COMMON"
          &DB2.      = "V8"
          &DCEPROC1. = "."
          &DFHSMCMD. = "00"
          &DFHSMHST. = "6"
          &DFHSMPRI. = "NO"
          &DFSPROC1. = "."
          &DLIB1.    = "Z17DL1"
          &DLIB2.    = "Z17DL2"
          &DLIB3.    = "Z17DL3"
          &DLIB4.    = "Z17DL4"
          &IEFSSNXX. = "R7"
          &IFAPRDXX. = "4A"
```

The IEASYMxx member of PARMLIB provides a single place to specify system parameters for each system in a multisystem environment. IEASYMxx contains statements that define static system symbols and specify IEASYSxx members that contain system parameters (the SYSPARM statement).

Example 1-5 shows a portion of a typical IEASYMxx member

Example 1-5 Partial IEASYMxx PARMLIB member (some lines removed)

```
SYSDEF          SYSCLONE(&SYSNAME(3:2))
                  SYMDEF(&SYSR2='&SYSR1(1:5).2')
                  SYMDEF(&SYSR3='&SYSR1(1:5).3')
                  SYMDEF(&DLIB1='&SYSR1(1:3).DL1')
                  SYMDEF(&DLIB2='&SYSR1(1:3).DL2')
                  SYMDEF(&DLIB3='&SYSR1(1:3).DL3')
                  SYMDEF(&DLIB4='&SYSR1(1:3).DL4')
                  SYMDEF(&ALLCLST1='CANCEL')
                  SYMDEF(&CMDLIST1='&SYSCLONE.,00')
```

```
                SYMDEF(&COMMDSN1='COMMON')
                SYMDEF(&DFHSMCMD='00')
                SYMDEF(&IFAPRDXX='00')
                SYMDEF(&DCEPROC1='.')
                SYMDEF(&DFSPROC1='.')
SYSDEF          HWNAME(SCZP901)
                LPARNAME(A13)
                SYSNAME(SC70)
                SYSPARM(R3,70)
                SYMDEF(&IFAPRDXX='4A')
                SYMDEF(&DFHSMHST='6')
                SYMDEF(&DFHSMPRI='NO')
                SYMDEF(&DB2='V8')
```

In the example, the variable &SYSNAME will have the value specified by the SYSNAME keyword - SC70 in this case. Because each system in a sysplex has a unique name, we can use &SYSNAME in the specification of system-unique resources, where permitted. As an example, we could specify the name of an SMF data set as SYS1.&SYSNAME..MAN1, with substitution resulting in the name SYS1.SC70.MAN1 when running on SC70.

You can use variables to construct the values of other variables. In Figure 1-5, we see &SYSCLONE taking on the value of &SYSNAME beginning at position 3 for a length of 2. Here, &SYSCLONE will have a value of 70. Similarly, we see &SYSR2 constructed from the first 5 positions of &SYSR1 with a suffix of 2. Where is &SYSR1 defined? &SYSR1 is system-defined with the VOLSER of the IPL volume. If you refer back to Figure 1-4 on page 20, you will see the values of &SYSR1 and &SYSR2.

We also see here the definition of a global variable defined to all systems - &IFAPRDXX with a value of 00 - and its redefinition for SC70 to a value of 4A.

System symbols are used in cases where multiple z/OS systems will share a single PARMLIB. Here, the use of symbols allows individual members to be used with symbolic substitution, as opposed to having each system require a unique member. The LOADxx member specifies the IEASYMxx member that the system is to use.

1.4 Managing system performance

The task of “tuning” a system is an iterative and continuous process, and it is the discipline that most directly impacts all users of system resources in an enterprise. The z/OS Workload Management (WLM) component is an important part of this process and includes initial tuning of selecting appropriate parameters for various system components and subsystems.

After the system is operational and criteria have been established for the selection of jobs for execution through job classes and priorities, WLM controls the distribution of available resources according to the parameters specified by the installation.

WLM, however, can only deal with available resources. If these are inadequate to meet the needs of the installation, even optimal distribution may 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 system programmer needs to modify WLM parameters to reflect changed conditions.

1.5 Configuring I/O devices

The I/O configuration to the operating system (software) and the channel subsystem (hardware) must be defined. The Hardware Configuration Definition (HCD) component of z/OS consolidates the hardware and software I/O configuration processes under a single interactive end-user interface (a large set of ISPF panels).

HCD is used for several purposes:

- Using input from the ISPF panels, it builds a special VSAM data set that defines the I/O configuration available to z/OS. The data set is known as an I/O definition file (IODF), which contains I/O configuration data. An IODF is used to define multiple hardware and software configurations to the z/OS operating system. When z/OS is started, an IODF must be specified. Every I/O device used by z/OS must be defined in the IODF.
- HCD builds an I/O definition for the complete mainframe machine and sends it to the controller section of the mainframe. This definition also specifies the existence of LPARs and other system-wide parameters. This is known as an I/O Configuration Data Set (IOCDS). The IOCDS is discussed in more detail in Chapter 3, “Hardware systems and LPARs” on page 77.
- HCD can be used to create a new IODF from an existing IODF and dynamically activate the new version.

When a new IODF is activated, HCD defines the I/O configuration to the channel subsystem and/or the operating system. With the HCD activate function or the z/OS ACTIVATE operator command, changes can be made in the current configuration without having to initial program load (IPL) the software or power-on reset (POR) the hardware. Making changes while the system is running is known as *dynamic configuration* or *dynamic reconfiguration*.

An IODF and IOCDS can contain definitions for I/O devices that do not exist or are not currently attached. It need not contain definitions for all I/O devices, although the system

(for the IOCDS) or z/OS (for an IODF) cannot use a device that is not defined. HCD is included with all z/OS systems.

1.6 Following a process of change control

Data center management is typically held accountable for Service Level Agreements (SLAs), often through a specialist team of service managers. Change control mechanics and practices in a data center are implemented to ensure that SLAs are met.

The implementation of any change must be under the control of the Operations staff. When a change is introduced into a production environment that results in problems or instability, Operations staff are responsible for observing, reporting, and then managing the activities required to correct the problem or back out the change.

Although system programmers will normally raise and implement their own changes, sometimes changes are based on a request through the change management system. Any instructions for Operations or other groups would be in the change record, and the approval of each group is required.

Implementing business application changes would normally be handled by a production control analyst. Application changes will normally reside in test libraries, and an official request (with audit trail) would result in the programs in the test libraries being promoted to the production environment.

Procedures involved in the change must be circulated to all interested parties. When all parties consider the change description to be complete, then it is considered for implementation and either scheduled, deferred, or possibly rejected.

The factors that need to be considered when planning a change are:

- The benefits that will result from the change
- What will happen if the change is not done
- The resources required to implement the change
- The relative importance of the change request compared to others
- Any interdependency of change requests

All change involves risk. One of the advantages of the mainframe is the very high availability that it offers. All change must therefore be carefully controlled and managed. A high proportion of any system programmer's time is involved in the planning and risk assessment of change. One of the most important aspects of change is how to reverse it and go back to the previous state.

1.6.1 Risk assessment

It is common practice for data center management to have a weekly change control meeting to discuss, approve, or reject changes. These changes might be for applications, a system, a network, hardware, or power.

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 may be permitted during the day, while higher risk changes would be scheduled for an outage slot.

It is also common practice for a data center to have periods of low and high risk, which will influence decisions. For example, if the system runs credit authorizations, then the periods around major public holidays are usually extremely busy and may cause a change freeze. Also, annual sales are extremely busy periods in retailing and may cause changes to be rejected.

IT organizations achieve their goals through disciplined change management processes and policy enforcement. These goals include:

- High service availability
- Increased security
- Audit readiness
- Cost savings

1.6.2 Change control record system

A *change control record system* is typically in place to allow for the requesting, tracking, and approval of changes. This is usually the partner of a *problem management system*. For example, if a production system has a serious problem on a Monday morning, then one of the first actions will be to examine the changes that were implemented over the weekend to determine if these have any bearing on the problem.

These records also show that the system is under control, which is often necessary to prove to auditors, especially in the heavily regulated financial services sector. The Sarbanes-Oxley Act of 2002 in the United States, which addresses corporate governance, has established the need for an effective internal control system. Demonstrating strong change management and problem management in IT services is part of compliance with this measure. Additionally, the 8th Directive on Company Law in the European Union, which is under discussion at the time of writing, will address similar areas to Sarbanes-Oxley.

For these reasons, and at a bare minimum, before any change is implemented there should be a set of controlled documents defined, which are known as *change request forms*. These should include the following:

- Who - that is, the department, group or person that requires the change, who is responsible for implementing the change, completing the successful test and responsible for backout if required. Also who will “sign off” the change as successful.
- What - that is, the affected systems or services (for example e-mail, file service, domain, etc.). Include as much detail as possible. Ideally, complete instructions should be included so that the change could be performed by someone else in an emergency.
- Where - that is, scope of change, the business units, buildings, departments or groups affected or required to assist with the change.
- When - that is, start date and time and estimated duration of the change. There are often three dates: requested, scheduled. and actual.
- Priority - that is, high, medium, low, business as usual, emergency, dated (for example clock change).
- Risk - that is, high, medium, low
- Impact - that is, what will happen if the change is implemented; what will happen if it is not; what other systems may be affected; what will happen if something unexpected occurs.

1.6.3 Production control

Production control usually involves a specialized staff to manage batch scheduling, using a tool such as Tivoli® Workload Scheduler to build and manage a complex batch schedule. This work might involve daily and weekly backups running at particular points within a complex sequence of application suites. Databases and online services might also be taken down and brought back up as part of the schedule. While making such changes, production control often needs to accommodate public holidays and other special events such as (in the case of a retail sales business) a winter sale.

Production control is also responsible for taking a programmer’s latest program and releasing it to production. This task typically involves moving the source code to a secure production library, recompiling the code to produce a production load module, and placing that module in a production load library. JCL is copied and updated to production standards and placed in appropriate procedure libraries, and application suites added to the job scheduler.

There might also be an interaction with the system programmer if a new library needs to be added to the linklist, or authorized.

1.7 Configuring consoles

Operating z/OS involves managing hardware such as processors and peripheral devices (including the consoles where your operators do their work); and software such as the z/OS operating control system, the job entry subsystem, subsystems (such as NetView®) that can control automated operations, and all the applications that run on z/OS.

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

- Message and command processing that forms the basis of operator interaction with z/OS and the basis of z/OS automation
- Console operations, or how operators interact with z/OS to monitor or control the hardware and software

Planning z/OS operations for a system must take into account how operators use consoles to do their work and how to manage messages and commands. The system programmer needs to ensure that operators receive the necessary messages at their consoles to perform their tasks, and select the proper messages for suppression, automation, or other kinds of message processing.

In terms of z/OS operations, how the installation establishes console recovery or whether an operator must re-IPL a system to change processing options are important planning considerations.

Because messages are also the basis for automated operations, the system programmer needs to understand message processing to plan z/OS automation.

As more installations make use of multisystem environments, the need to coordinate the operating activities of those systems becomes crucial. Even for single z/OS systems, an installation needs to think about 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 enter from consoles can be a security concern that requires careful coordination. As a planner, the system programmer needs to make sure that the right people are doing the right tasks when they interact with z/OS.

A *console configuration* consists of the various consoles that operators use to communicate with z/OS. Your installation first defines the I/O devices it can use as consoles through the Hardware Configuration Definition (HCD), an interactive interface on the host that allows the system programmer to define the hardware configuration for both the channel subsystem and operating system.

Hardware Configuration Manager (HCM) is the graphical user interface to HCD. HCM interacts with HCD in a client/server relationship (that is, HCM runs on a workstation and HCD runs on the host). The host systems require an internal model of their connections to devices, but it can be more convenient and efficient for the system

programmer to maintain (and supplement) that model in a visual form. HCM maintains the configuration data as a diagram in a file on the workstation in sync with the IODF on the host. While it is possible to use HCD directly for hardware configuration tasks, many customers prefer to use HCM exclusively, due to its graphical interface.

Besides HCD, once the devices have been defined, z/OS is told which devices to use as consoles by specifying the appropriate device numbers in the CONSOLxx PARMLIB member.

Generally, operators on a z/OS system receive messages and enter commands on MCS and SMCS consoles. They can use other consoles (such as NetView consoles) to interact with z/OS, but here we describe MCS, SMCS, and EMCS consoles and how to plan for their use:

- *MCS consoles* are devices that are locally attached to a z/OS system and provide the basic communication between operators and z/OS. (MCS consoles are attached to control devices that do *not* support systems network architecture (SNA) protocols.)
- *SMCS consoles* are devices that do not have to be locally attached to a z/OS system and provide the basic communication between operators and z/OS. SMCS consoles use z/OS Communications Server to provide communication between operators and z/OS, instead of direct I/O to the console device.
- *EMCS consoles* are devices (other than MCS or SMCS consoles) from which operators or programs can enter commands and receive messages. Defining extended MCS consoles as part of the console configuration allows the system programmer to extend the number of consoles beyond the MCS console limit, which is 99 for an z/OS system or sysplex.

The system programmer defines these consoles in a configuration according to their functions. For example, one console can function as a master console for the system. Important messages that require action can be directed to the operator, who can act by entering commands on the console. Another console can act as a monitor to display messages to an operator working in a functional area like a tape pool library, or to display messages about printers at your installation.

Figure 1-5 shows a console configuration for a z/OS system that also includes the system console, an SMCS console, NetView, and TSO/E.

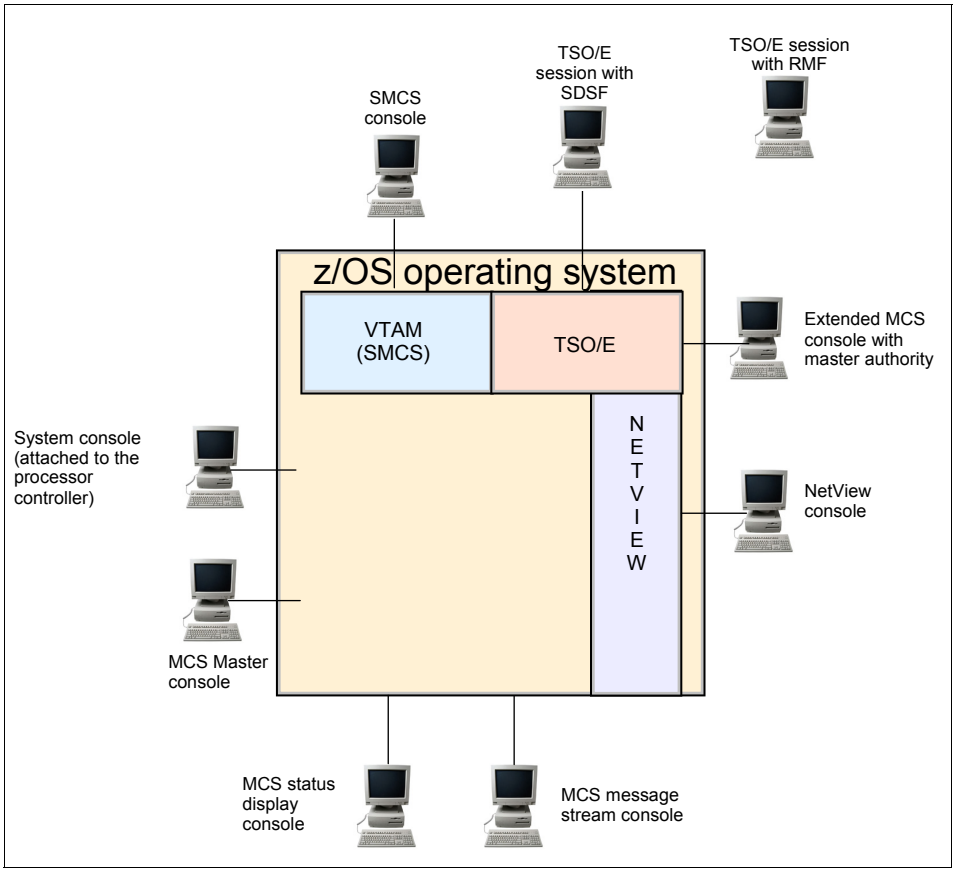


Figure 1-5 Sample console configuration for a z/OS system

The *system console* function is provided as part of the Hardware Management Console (HMC). An operator can use the system console to initialize z/OS and other system software, and during recovery situations when other consoles are unavailable.

Besides MCS and SMCS consoles, the z/OS system shown in Figure 1-5 has a NetView console defined to it. NetView works with system messages and command lists to help automate z/OS operator tasks. Many system operations can be controlled from a NetView console.

Users can monitor many z/OS system functions from TSO/E terminals. Using the System Display and Search Facility (SDSF) and the Resource Measurement Facility (RMF™), TSO/E users can monitor z/OS and respond to workload balancing and performance problems. An authorized TSO/E user can also initiate an extended MCS console session to interact with z/OS.

The MCS consoles shown in Figure 1-5 are:

- An MCS master console from which an operator can view messages and enter all z/OS commands

  This console is in full capability mode because it can receive messages and accept commands. An operator can control the operations for the z/OS system from an MCS or SMCS master console.

- An MCS status display console

  An operator can view system status information from DEVSERV, DISPLAY, TRACK, or CONFIG commands. However, because this is a status display console, an operator cannot enter commands from the console. An operator on a full capability console can enter these commands and route the output to a status display console for viewing.
  Note: an SMCS console cannot be a status display console.

- An MCS message-stream console

  A message-stream console can display system messages. An operator can view messages routed to this console. However, because this is a message-stream console, an operator cannot enter commands from here. Routing codes and message level information for the console are defined so that the system can direct relevant messages to the console screen for display. Thus, an operator who is responsible for a functional area like a tape pool library, for example, can view MOUNT messages. An SMCS console cannot be a message stream console.

In many installations, this proliferation of screens has been replaced by operator workstations that combine many of these screens onto one windowed display. Generally, the hardware console is separate, but most other terminals are combined. The systems are managed by alerts for exception conditions from the automation product.

The IBM Open Systems Adapter-Express Integrated Console Controller (OSA-ICC) is the modern way of connecting consoles. OSA-ICC uses TCP/IP connections over Ethernet LAN to attach to personal computers as consoles through a TN3270 connection (telnet).

1.8 Using SMP/E to manage changes

System Management Program/Extended (SMP/E) is the program used by the system programmers to install software products, updates, and fixes on z/OS. Using SMP/E is one of the most complex functions associated with maintaining z/OS software. Figure 0-1 illustrates the general flow when using SMP/E to apply a fix to z/OS. Such fixes for z/OS software are known as PTFs, for Product Temporary Fix.

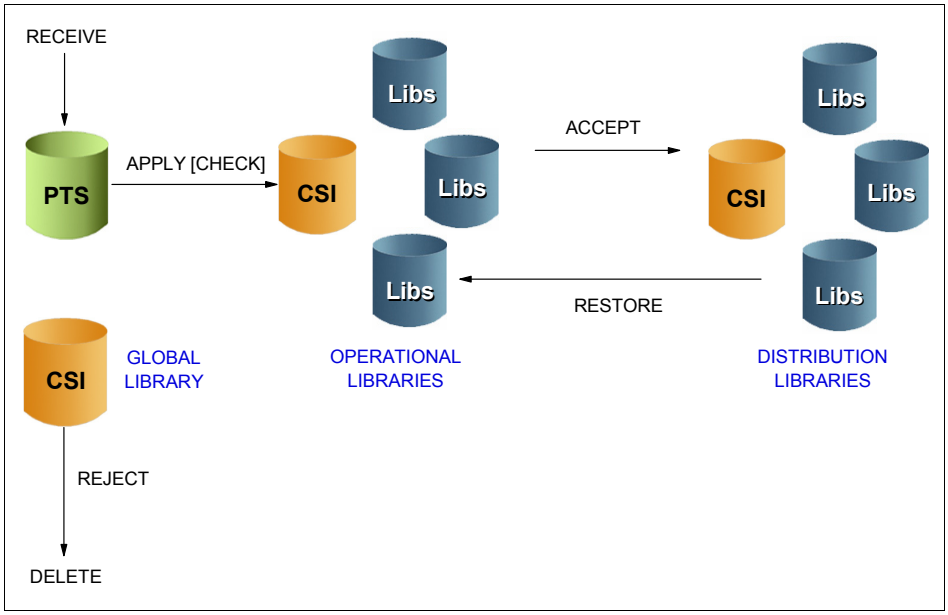


Figure 0-1 SMP/E overview for fixes

SMP/E works with three levels of libraries:

- A receive-level library that is a holding area - This library is known as the PTS (PTF Temporary Storage) and is in the *GLOBAL zone*.
- Operational or *target libraries* - These are the working libraries used by z/OS and various software products while they are installed for use.
- Distribution libraries or *DLIBs* - These contain a master copy of all the modules in the operational libraries. The DLIBs are not used for execution.

Each set of libraries has a Consolidated Software Index (CSI), which is a VSAM data set containing data about all the modules in the set. In principle, it is possible to rebuild any operational z/OS module from the DLIBs. The CSI contains all the binder (linkage editor) control statements needed to rebuild an operational module from submodules in the DLIBs.

The SMP/E RECEIVE command is used to accept modules (or submodules of a larger module) that are fixes for distributed products. The RECEIVEed module is placed in the PTS. The APPLY CHECK command is used to determine whether the fix can be cleanly applied to the operational module. (This may involve relinking/rebinding many submodules.) If these checks are positive, the APPLY command is used to install the fix in the operational libraries. The installation would then use the fixed module and determine whether it is acceptable. If it is not acceptable, a RESTORE command can rebuild the original module from material in the DLIBs.

If the fix is acceptable (and it make take months to determine this) an ACCEPT command integrates the fix module into the DLIBs. After this is done, the fix can no longer be *backed out* of the system (except by restoring old backup tapes, perhaps). Future fixes can then be built on top of this fix in the DLIBs. A REJECT command is used to discard a fix from the global area; this might be done if someone decides the fix will never be used.

SMP/E control statements can verify the status of previous and concurrent fixes, the existence of a product and its release level, and many other factors.

z/OS and practically all z/OS software products from IBM are installed using SMP/E. Most independent software vendor products are installed by using SMP/E. Many z/OS installations will not purchase any software that is not installed with SMP/E.

Each product can have its own global, target, and DLIB libraries, or many products can be placed in the same set of global, target, and DLIB libraries. SMP/E is included with all z/OS systems.

SMP/E usage can be quite complex and SMP/E skills are a primary prerequisite for a z/OS system programmer. The DLIBs for a product (including z/OS) tend to be as large as the operational libraries. In a sense, almost all the software for z/OS and most associated software products are present twice on disk, once for the operational libraries and once for the distribution libraries.

In earlier days, when disks were smaller and more expensive, some installations did not keep their DLIBs online. In principle, they are needed only when a fix or new product is installed and they could occupy a number of disk drives. Smaller installations might dump the DLIB volumes onto tape and restore them (to scratch disks) only when needed. In even earlier days the use of SMP/E was considered optional and some installations did not maintain their DLIBs. In today's systems, however, the DLIBs are required for most practical purposes and are usually available online.

1.9 Initializing the system

An initial program load (IPL) is the act of loading a copy of the operating system from disk into the processor's real storage and executing it.

z/OS systems are designed to run continuously with many months between reloads, allowing important production workloads to be continuously available. Change is the usual reason for a reload, and the level of change on a system dictates the reload schedule. For example:

- A test system may be IPLed daily or even more often.
- A high-availability banking system may only be reloaded once a year, or even less frequently, to refresh the software levels.

- Outside influences may often be the cause of IPLs, such as the need to test and maintain the power systems in the machine room.
- Sometimes badly behaved software uses up system resources that can only be replenished by an IPL, but this sort of behavior is normally the subject of investigation and correction.

Many of the changes that required an IPL in the past can now be done dynamically. Examples of these tasks are:

- Changing the content of linklist, the APF list or the IO configuration
- Adding modules to LPA

z/OS is IPLed using the Hardware Management Console (HMC). You need to supply the following information to IPL z/OS:

- Device address of the IPL volume
- Suffix of the LOADxx member, which contains pointers to system parameters and the IODF data set, which contains the configuration information.
- Device address of the IODF volume, which is the starting point for the system search for the SYSn.IPLPARM data set that contains the LOADxx member.
- Message suppression character
- Optional nucleus suffix

1.9.1 Initialization process

The system initialization process (Figure 1-6 on page 33) prepares the system control program and its environment to do work for the installation. This process essentially consists of:

- System and storage initialization, including the creation of system component address spaces
- Master scheduler initialization and subsystem initialization

When the system is initialized and the job entry subsystem is active, the installation can submit jobs for processing by using the START, LOGON, or MOUNT command.

The initialization process begins when the system programmer selects the LOAD function at the Hardware Management Console (HMC). z/OS locates all usable central storage that is online and available, and begins creating the various system areas.

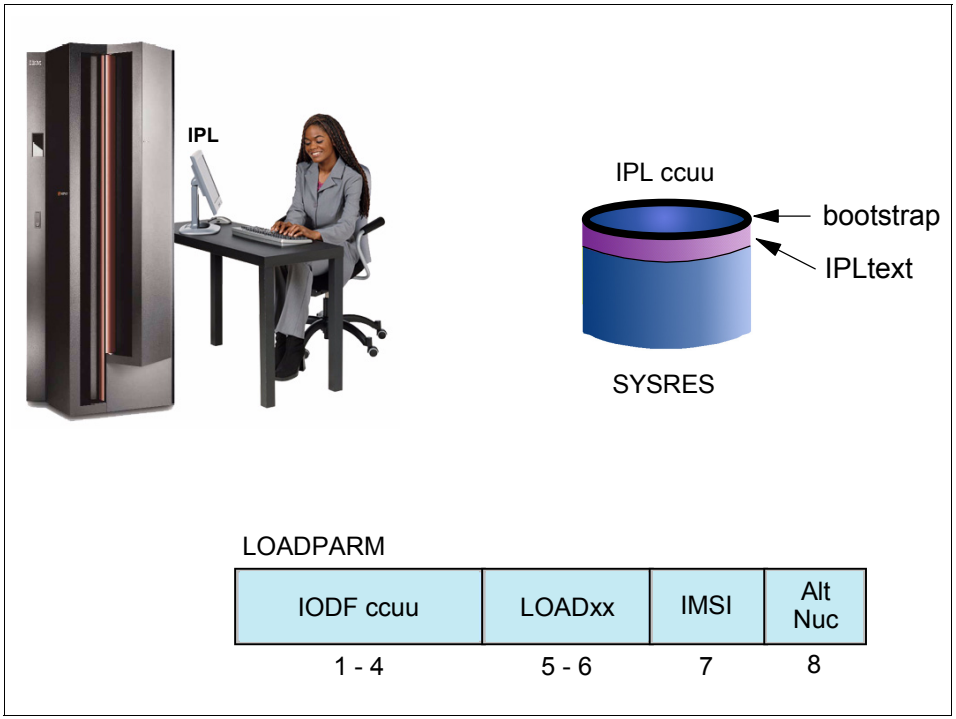


Figure 1-6 IPLing the machine

Not all disks attached to a CPU have loadable code on them. A disk that does is generally referred to as an “IPLable” disk, and more specifically as the SYSRES volume.

IPLable disks contain a bootstrap module at cylinder 0 track 0. At IPL, this bootstrap is loaded into storage at real address zero and control is passed to it. The bootstrap then reads the IPL control program IEAIPL00 (also known as IPL text) and passes control to it. This in turn starts the more complex task of loading the operating system and executing it.

After the bootstrap is loaded and control is passed to IEAIPL00, IEAIPL00 prepares an environment suitable for starting the programs and modules that make up the operating system, as follows:

1. It clears central storage to zeros before defining storage areas for the master scheduler.
2. It locates the SYS1.NUCLEUS data set on the SYSRES volume and loads a series of programs from it known as IPL Resource Initialization Modules (IRIMs).
3. These IRIMs begin creating the normal operating system environment of control blocks and subsystems.

Some of the more significant tasks performed by the IRIMs are as follows:

- Read the LOADPARM information entered on the hardware console at the time the IPL command was executed.
- Search the volume specified in the LOADPARM member for the IODF data set. IRIM will first attempt to locate LOADxx in SYS0.IPLPARM. If this is unsuccessful, it will look for SYS1.IPLPARM, and so on, up to and including SYS9.IPLPARM. If at this point it still has not been located, the search continues in SYS1.PARMLIB. (If LOADxx cannot be located, the system loads a wait state.)
- If a LOADxx member is found, it is opened and information including the nucleus suffix (unless overridden in LOADPARM), the master catalog name, and the suffix of the IEASYSxx member to be used, is read from it.
- Load the operating system's nucleus.
- Initialize virtual storage in the master scheduler address space for the System Queue Area (SQA), the Extended SQA (ESQA), the Local SQA (LSQA), and the Prefixed Save Area (PSA). At the end of the IPL sequence, the PSA will replace IEAIPL00 at real storage location zero, where it will then stay.
- Initialize real storage management, including the segment table for the master scheduler, segment table entries for common storage areas, and the page frame table.

The last of the IRIMs then loads the first part of the Nucleus Initialization Program (NIP), which invokes the Resource Initialization Modules (RIMs), one of the earliest of which starts up communications with the NIP console defined in the IODF.

The system continues the initialization process, interpreting and acting on the system parameters that were specified. NIP carries out the following major initialization functions:

- Expands the SQA and the extended SQA by the amounts specified on the SQA system parameter.
- Creates the pageable link pack area (PLPA) and the extended PLPA for a cold start IPL; resets tables to match an existing PLPA and extended PLPA for a quick start or a warm start IPL.
- Loads modules into the fixed link pack area (FLPA) or the extended FLPA. Note that NIP carries out this function only if the FIX system parameter is specified.
- Loads modules into the modified link pack area (MLPA) and the extended MLPA. Note that NIP carries out this function only if the MLPA system parameter is specified.
- Allocates virtual storage for the common service area (CSA) and the extended CSA. The amount of storage allocated depends on the values specified on the CSA system parameter at IPL.
- Page-protects the NUCMAP, PLPA and extended PLPA, MLPA and extended MLPA, FLPA and extended FLPA, and portions of the nucleus.

Note: An installation can override page protection of the MLPA and FLPA by specifying NOPROT on the MLPA and FIX system parameters.

Related Reading: For more information about quick starts and warm starts, see the IBM publication *z/OS MVS Initialization and Tuning Reference*.

IEASYSnn, a member of PARMLIB, contains parameters and pointers that control the direction that the IPL takes. See Example 1-7:

```
------------------------------------------------------------------------------
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       SYS1.PARMLIB(IEASYS00) - 01.68                Columns 00001 00072
Command ===>                                               Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 ALLOC=00,
000002 APG=07,
000003 CLOCK=00,
000004 CLPA,
000005 CMB=(UNITR,COMM,GRAPH,CHRDR),
000006 CMD=(&CMDLIST1.),
000007 CON=00,
000008 COUPLE=00,  WAS FK
000009 CSA=(2M,128M),
000010 DEVSUP=00,
000011 DIAG=00,
000012 DUMP=DASD,
000013 FIX=00,
000014 GRS=STAR,
000015 GRSCNF=ML,
000016 GRSRNL=02,
000017 IOS=00,
000018 LNKAUTH=LNKLST,
000019 LOGCLS=L,
000020 LOGLMT=999999,
000021 LOGREC=SYS1.&SYSNAME..LOGREC,
000022 LPA=(00,L),
000023 MAXUSER=1000,
000024 MSTRJCL=00,
000025 NSYSLX=250,
000026 OMVS=&OMVSPARM.,
------------------------------------------------------------------------------------------------------------
```

Figure 1-7 Partial listing of IEASYS00 member

To see information about how your system was IPLed, you can enter the command **D IPLINFO** as shown in Example 1-6.

Example 1-6 Output of D IPLINFO command

```
D IPLINFO
IEE254I  11.11.35 IPLINFO DISPLAY 906
 SYSTEM IPLED AT 10.53.04 ON 08/15/2005
 RELEASE z/OS 01.07.00    LICENSE = z/OS
 USED LOADS8 IN SYS0.IPLPARM ON C730
 ARCHLVL = 2   MTLSHARE = N
 IEASYM LIST = XX
 IEASYS LIST = (R3,65) (OP)
 IODF DEVICE C730
 IPL DEVICE 8603 VOLUME Z17RC1
```

System address space creation

In addition to initializing system areas, z/OS establishes system component address spaces. It establishes an address space for the master scheduler and other system address spaces for various subsystems and system components. Some of the component address spaces are: *MASTER*, ALLOCAS, APPC, CATALOG, and so on.

Master scheduler initialization

Master scheduler initialization routines initialize system services such as the system log and communications task, and start the master scheduler itself. They also cause creation of the system address space for the job entry subsystem (JES2 or JES3), and then start the job entry subsystem.

Subsystem initialization

Subsystem initialization is the process of readying a subsystem for use in the system. One or more IEFSSNxx members of PARMLIB contain the definitions for the primary subsystems such as JES2 or JES3, and the secondary subsystems such as NetView and DB2.

During system initialization, the defined subsystems are initialized. The primary subsystem (JES) is usually defined first because other subsystems, such as DB2, require the services of the primary subsystem in their initialization routines.

After the primary JES is initialized, the other subsystems are initialized in the order in which the IEFSSNxx members are specified by the SSN parameter. For example, for SSN=(aa,bb), PARMLIB member IEFSSNaa would be processed before IEFSSNbb.

START/LOGON/MOUNT processing

After the system is initialized and the job entry subsystem is active, jobs can be submitted for processing. When a job is activated through START (for batch jobs), LOGON (for time-sharing jobs), or MOUNT, a new address space is allocated. Note that before

LOGON, the operator must have started VTAM and TSO, which have their own address spaces.

Figure 1-8 shows some of the important system address spaces and VTAM, CICS, TSO, a TSO user and a batch initiator. Each address space has 2 GB of virtual storage by default, whether the system is running in 31-bit or 64-bit mode.

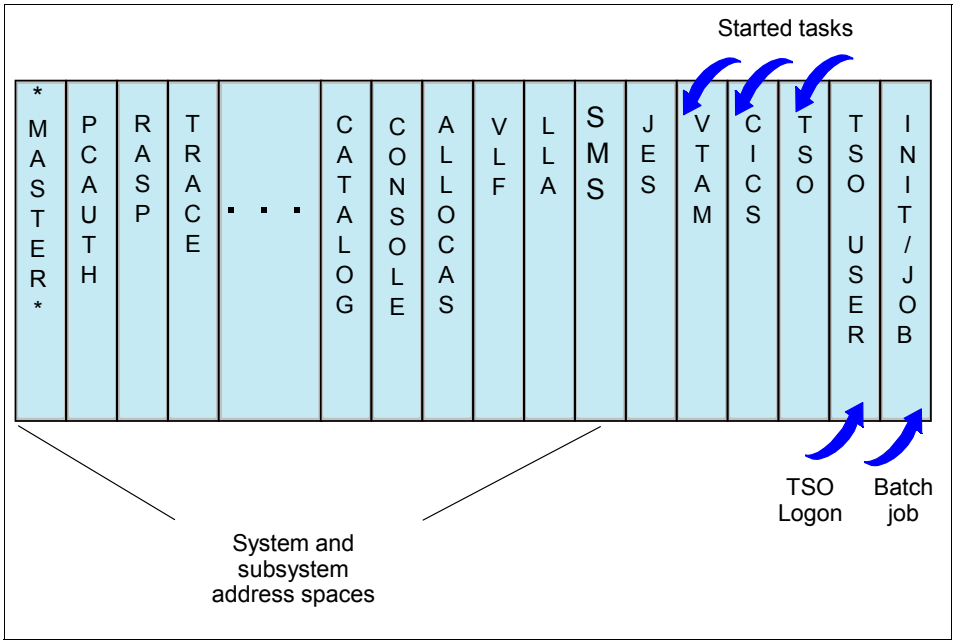


Figure 1-8 Virtual storage layout for multiple address spaces

Recall that each address space is divided into various common and private storage areas. The private areas are available only to that address space, but common areas are available to all.

During initialization of a z/OS system, the operator uses the system console or hardware management console, which is connected to the support element. From the system console, the operator initializes the system control program during the Nucleus Initialization Program (NIP) stage.

During the NIP stage, the system might prompt the operator to provide system parameters that control the operation of z/OS. The system also issues informational messages that inform the operator about the stages of the initialization process.

1.9.2 IPL types

Several types of IPL exist; these are described as follows:

- Cold start

  An IPL that loads (or reloads) the PLPA and clears the VIO data set pages. The first IPL after system installation is always a cold start because the PLPA is initially loaded. Subsequent IPLs are cold starts when the PLPA is reloaded, either to alter its contents or to restore its contents if they were lost. This is usually done when changes have been made to the LPA (for example, when a new SYSRES containing maintenance is being loaded).

- Quick start

  An IPL that does not reload the PLPA, but clears the VIO data set pages. (The system resets the page and segment tables to match the last-created PLPA.) This is usually done when there have been no changes to LPA, but VIO must be refreshed. This prevents the warm start of jobs that were using VIO data sets.

- Warm start

  An IPL that does not reload the PLPA, and preserves journaled VIO data set pages. This will allow jobs that were running at the time of the IPL to restart with their journaled VIO data sets.

Note: VIO is a method of using memory to store small temporary data sets for rapid access. However, unlike a RAM disk on a PC, these are actually backed up to disk and so can be used as a restart point. Obviously there should not be too much data stored in this way, so the size is restricted.

Often, the preferred approach is to do a cold start IPL (specifying CLPA). The other options can be used, but extreme care must be taken to avoid unexpected change or backout of change. A warm start could be used when you have long-running jobs which you want to restart after IPL, but an alternative approach is to break down those jobs into smaller pieces which pass real data sets rather than use VIO. Modern disk controllers with large cache memory have reduced the need for VIO data to be kept for long periods.

Also, do not confuse a cold start IPL (CLPA would normally be used rather than the term “cold start”) with a JES cold start. Cold starting JES is something that is done extremely rarely, if ever, on a production system, and totally destroys the existing data in JES.

1.9.3 Shutting down the system

To shut down the system each task must be closed in turn, in the correct order. On modern systems, this is the task of the automation package. Shutting down the system usually requires a single command. This will remove most tasks except Automation itself. The Automation task is closed manually, and then any commands needed to remove the system from a sysplex or serialization ring are issued.

1.10 Summary

The role of the z/OS system programmer is to install, customize, and maintain the operating system.

The system programmer must understand the following areas (and more):

- System customization
- Workload management
- System performance
- I/O device configuration
- Operations

To maximize the performance of the task of retrieving modules, the z/OS operating system has been designed to maintain in memory those modules that are needed for fast response to the operating system, as well as for critical applications. Link pack area (LPA), linklist, and authorized libraries are the cornerstones of the fetching process.

Also discussed was the system programmer's role in configuring consoles, setting up message-based automation, and using SMP/E to manages changes to the system software.

System start-up or IPL was introduced with the following topics:

- IPL and the initialization process
- Types of IPLs: cold start, quick start, and warm start
- Reasons for IPLing

Key terms in this section		
HCD	IODF	SYSRES
SMP/E	linklist	IPL
WTOR	PARMLIB	PROCLIB
system symbol	PSA	LPA
nucleus	LOADPARM	SQA
system library		

2

Using SMP/E

Objective: As a z/OS system programmer, you will need to manage changes to the system software. You must ensure that software products and their modifications, service levels or fixes are properly installed at a specific level so that their interfaces and elements can work together with the operating system. As more products integrate and the system is enhanced, so, too, does the complexity of monitoring the elements of the system increase. This is an extremely vital and critical role.

After completing this topic, you will be able to:

- Explain how proper installation and maintenance are the basis for high availability in the z/OS environment.

2.1 What is SMP/E?

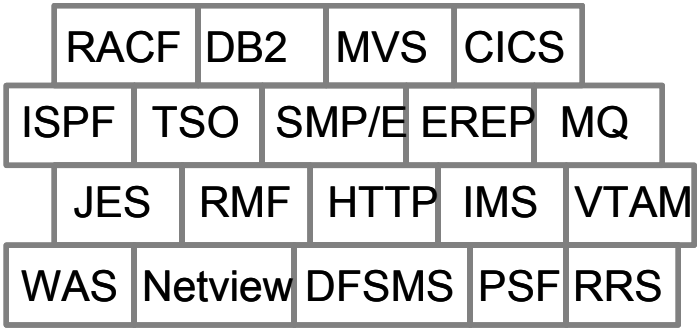
SMP/E is the basic tool for installing and maintaining software on z/OS and its related subsystems and middleware. It controls these changes at the component level by:

- Selecting the proper level of code to be installed from a large number of potential changes keeping the product software consistent
- 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) made up of one or more VSAM data sets.
- This function protects the integrity of the software.

SMP/E can be run either using batch jobs or using dialogues under Interactive System Productivity Facility/Program Development Facility (ISPF/PDF). SMP/E dialogs help you interactively query the SMP/E database, as well as create and submit jobs to process SMP/E commands.

2.2 The SMP/E view of the system

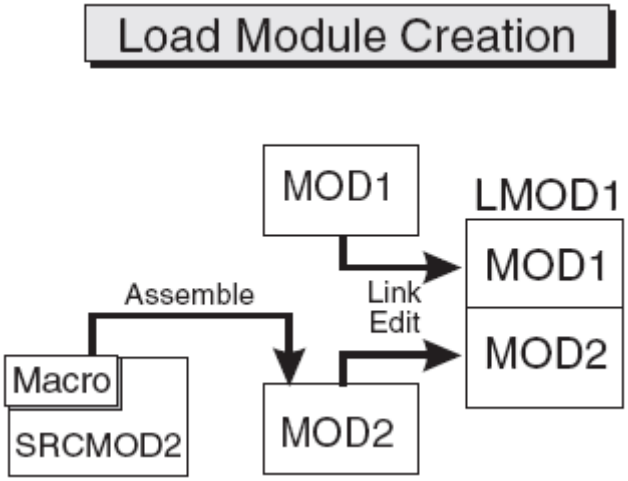
The z/OS system overall view may appear to be one big block of code that drives the CPU. Although, z/OS and previous versions are highly complex comprising many different smaller blocks of code. Each of those smaller blocks of code perform a specific function within the system.



Each system function is composed of one ore more load modules. In a z/OS environment, a load module represents the basic unit of machine-readable execution code (Operation Code). Load modules are created by combining one or more object modules and

processing them with a link-edit utility. The link-editing of modules is a process that resolves external references (called routines) and addresses. The functions on your system, therefore are one or more object modules that have been combined and link-edited.

To examine where the object module comes from, let's take a look at this example:



At most times, object modules are sent to you as part of a product. In the above example, the object module **MOD1** was sent as part of a product. Other times, you may need to assemble source code sent to you by the product packagers to create the object module. You can modify the source code and then assemble it to produce an object module. In this example **SRCMOD2** is source code that you assemble to create object module **MOD2**. When assembled, you link edit object module **MOD2** with object module **MOD1** to create the load module **LMOD1.**

In addition, to object modules and source code, most products distribute many additional parts such as macros, help-panels, CLISTs and other z/OS library members. These modules, macros and other types of data and code are the basic building blocks of your system. All of these building blocks are called *elements*. They also describes the relationship the software has with other products or services that may be installed on the same z/OS system.

2.3 Changing system elements

Over time, you may need to change some of the elements of your system. These changes may be necessary to improve the usability or reliability of a product. You may want to add some new functions to your system, upgrade some of the elements of the system, or modify some elements for a variety of reasons. In all cases, you are making system modifications. In SMP/E, we refer to these system modifications as *SYSMODs*.

A SYSMOD is the actual package containing information SMP/E needs to install and track system modifications. SYSMOD are composed of two parts:

- Modification Control Statement (MCS) designated by ++ as the first two characters, that tells SMP/E:
  - What elements are being updated or replaced
  - How the SYSMOD relates to product software and other SYSMODs
  - Other specific installation information
- The Modification Text which is the object modules, macros and other elements supplied by the SYSMOD.

There are four different categories of SYSMODs, each supporting a task you might want to perform:

- Function SYSMOD - Introduce the element for a product
- PTF SYSMOD (program temporary fix) - Prevent or fix problems with an element or introducing a new element.
- APAR SYSMOD (authorized program analysis report) - Fix problems with an element.
- USERMOD SYSMOD (user modification) - Customize an element.

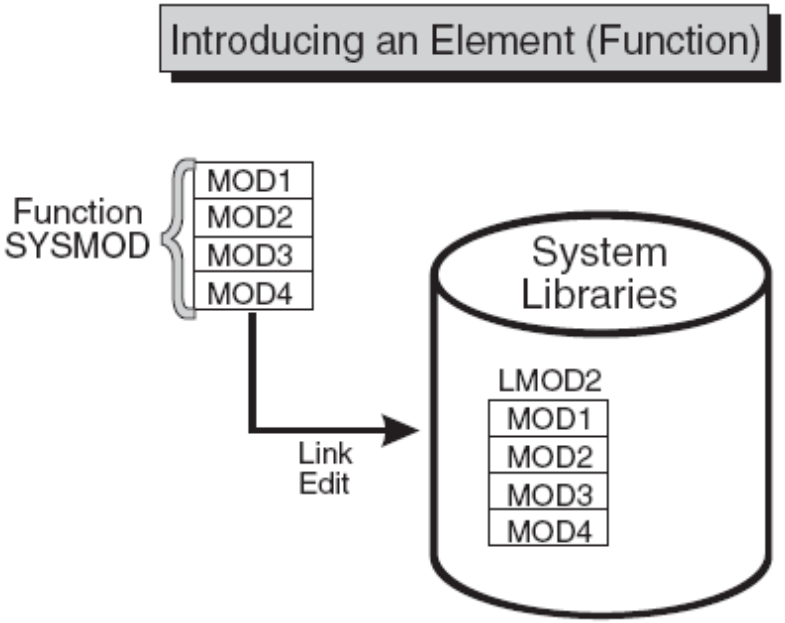
2.4 Introducing an element into the system

One way you can modify your system is to introduce new elements into that system. To accomplish this with SMP/E, you can install a function SYSMOD. The function SYMOD introduces a new product, a new version or release of a product, or updated functions for an existing product into the system.

Important: All other types of SYSMODs are dependent upon the function SYSMOD, because they are all modifications of the elements originally introduced by the function SYSMOD.

When we refer to installing a function SYSMOD, we are referring to the placing of all product's elements in the system data sets, or libraries. Examples of these libraries are SYS1.LINKLIB, SYS1.LPALIB and SVCLIB. The figure below displays the process of creating executable code in the production system libraries.

example



The above figure shows the installation of a function SYSMOD links-edits object modules MOD1, MOD2, MOD3 and MOD4 to create load module LMOD2. The executable code created in load module LMOD2 is installed in the system libraries through the installation of the function SYSMOD.

There are two types of function SYSMODs:

- A base function SYSMOD adds or replaces the entire system function
- A dependent function SYSMOD provides an addition to an existing system function. It is called dependent because its installation depends upon a base function already installed.

Both of these functions are used to introduce new elements into the system and below is an example of a simple function SYSMOD that introduces four elements:

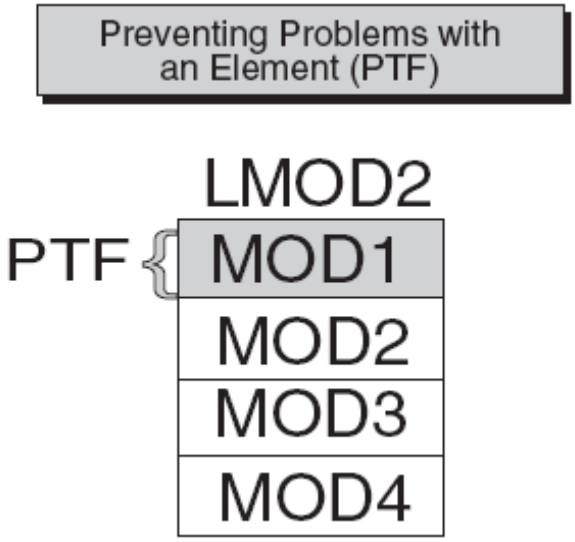
example

```
++FUNCTION(FUN0001)          /* SYSMOD type and identifier.   */.
++VER(Z038)                  /* For an OS/390 system          */.
++MOD(MOD1)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD2)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD3)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD4)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
```

2.5 Preventing problems with an element

When a problem with a software element is discovered, IBM supplies its customers with a tested fix for that problem. This will come in a form of a program temporary fix (PTF). Although you may not have experienced the problem the PTF is intended to prevent, it is wise to install the PTF on your system. The PTF SYSMOD is used to install the PTF, thereby preventing the occurrence of that problem on your system.

Most PTFs are designed to update or replace one or more complete elements of a system function. Here is a small example:



In the above diagram we see a previously installed load module, LMOD2. If we want to replace the element MOD1, we should install a PTF SYSMOD that contains the module

MOD1. That PTF SYSMOD replaces the element in error with the corrected element. As part of the installation of the PTF SYSMOD, SMP/E relinks LMOD2 to include the new and corrected version of MOD1.

Here is an example of a simple PTF SYSMOD:

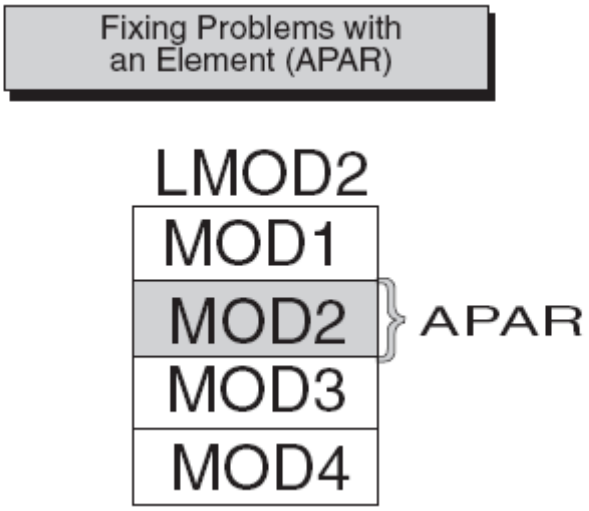
```
++PTF(PTF0001)              /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)   /* Apply to this product.        */.
++MOD(MOD1)                 /* Replace this module           */
            DISTLIB(AOSFB)  /* in this distribution library. */.
...
... object code for module
...
```

Note: From the above example we will describe Distribution Libraries later.

PTF SYSMODs are always dependent upon the installation of a function SYSMOD. In some cases, some PTF SYSMODs may also be dependent upon the installation of other PTF SYSMODs. These dependencies are called prerequisites. We will look at the typical PTF prerequisite when we discuss the complexity of keeping track of the elements of the system.

2.6 Fixing problems with an element

At times, you will find it necessary to correct a serious problem that occurs on your system before a PTF is ready for distribution. Therefore, in this circumstance, IBM supplies you with an authorized program analysis report (APAR). An APAR is a fix designed to correct a specific area of an element or replace an element in error. You install an APAR SYSMOD to implement a fix, therefore updating the incorrect element.



The processing of the APAR SYSMOD provides a modification for the object module MOD2. During the installation of the APAR SYSMOD, MOD2 is updated (and corrected) in LMOD2.

```
++APAR(APAR001)             /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)   /* Apply to this product         */
            PRE(UZ00004)    /* at this service level.        */.
++ZAP(MOD2)                 /* Update this module            */
            DISTLIB(AOSFB)  /* in this distribution library. */.
...
... zap control statements
...
```

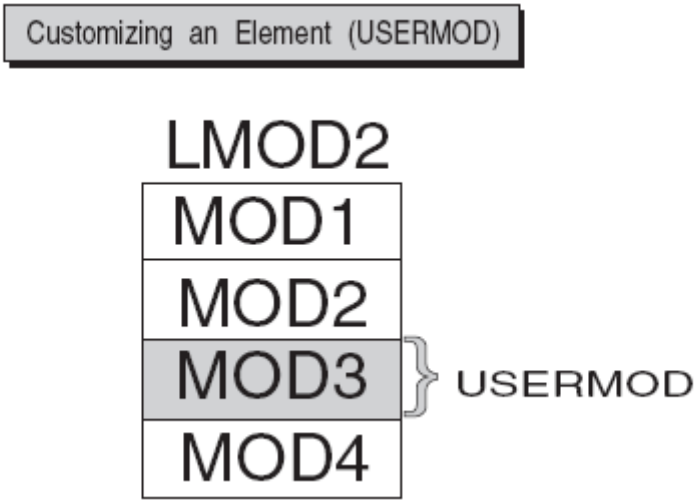
The APAR SYSMOD always has the installation of a function SYSMOD as a prerequisite and can also be dependent upon the installation of other PTFs or APAR SYSMODs.

2.7 Customizing an element - USERMOD SYSMOD

If you have a requirement for a product to perform differently than what it was intended to, you can consider a customized element for that system. IBM provides certain modules allowing you to adapt IBM code for your specific requirements. After you make the desired modifications, you place these modules into your system by installing a USERMOD SYSMOD. This SYSMOD can be used to replace or update an element or to introduce a totally new written element into your system. In all circumstances, the

USERMOD SYSMOD is created by you either to modify IBM code or to add your own code to the system.

Here we see that MOD3 has changed through a USERMOD SYSMOD.



Below is the example of a USERMOD SYSMOD

```
++USERMOD(USRMOD1)              /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)       /* Apply to this product         */
            PRE(UZ00004)        /* at this service level.        */.
++SRCUPD(JESMOD3)               /* Update this source module     */
            DISTLIB(AOSFB)      /* in this distribution library. */.
...
... update control statements
...
```

Note: For a USERMOD SYSMOD there might be a prerequisite for the installation of a function SYSMOD and possibly the installation of other APARs, PTFs or USERMOD SYSMODs.

2.7.1 Prerequisites and corequisites

Again, a function SYSMOD is a prerequisites for APARs, PTFs and USERMODs. Sometimes a PTF or even an APAR is dependent upon other PTF SYSMODs called

corequisites. Here we begin to see the complexity of elements and SYSMOD dependencies. In order to keep track of hundreds of modules in many system libraries we finally now understand the need for SMP/E tooling. Keeping track of these components and changes become very apparent when we examine the z/OS maintenance process since you can see how a PTF can contain multiple elements to replace.

2.8 Tracking and controlling elements

The means for SMP/E to track and control elements successfully is to identify them uniquely. This is where we use the *modification identifiers* and there are three types.

- Function Modification Identifiers (FMIDs) identify the function SYSMOD that introduces the element into the system.
- Replacement Modification Identifiers (RMIDs) identify the last SYSMOD (in most cases a PTF SYSMOD) to replace an element.
- Update Modification Identifiers (UMIDs) which identifies the SYSMOD that an update to an element since it was last replaced.

2.9 Working with SMP/E

The SMP/E process is performed by three simple basic commands, (RECEIVE, APPLY, ACCCEPT). Before we explore these command we will look at Libraries that are used to implement the process. There are two types of libraries. The Distribution and Target Libraries.

The Distribution Libraries (DLIBs) contains all the elements used for input and is also used for backup. In the case of an error with a Production element, the element can be replaced from these libraries.

The Target Libraries (TLIBs) contain all the executables required to run the system.

As mentioned earlier the Consolidated Software Inventory (CSI) is where SMP/E keeps track of SYSMODs composed of one or more VSAM KSDS data sets. These are divided into zones that contain groupings of records or their installation status describing the SYSMOD and their associated elements called *entities*.

The elements found in the distribution libraries are contained in the distribution zone and entry elements found in the target libraries are contained in the target zone. In addition to these zones the CSI contains a global zone.

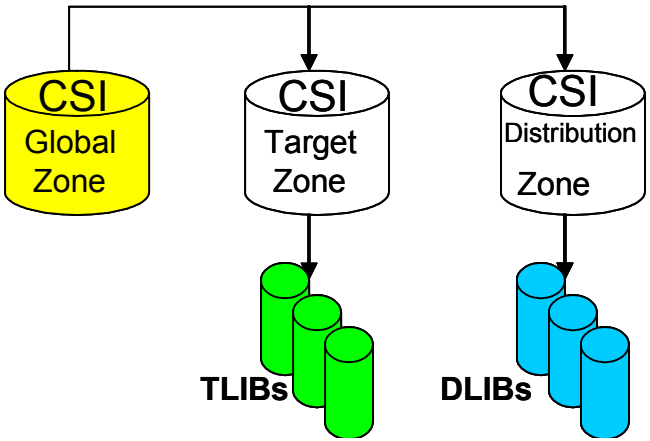
The Global Zone provides:

- The options found for SMP/E processing

- Status information for all SYSMODs in process
- Entries required to identify and describe each Target and Distribution zone
- Special exception handling for SYSMOD in error.

  The exception data handling is very important. It is referred to as *HOLDDATA* and is usually supplied for a product to indicate that the SYSMOD should not be processed or held for installation. In these cases the cause can be:

  - Performing some action before installing the SYSMOD (USER HOLD)
  - The PTF is in error and awaiting correction (ERROR HOLD)
  - The system needs to perform an action before it is installed (SYSTEM HOLD)



All the information located in the Global Zone, combined with the information found in the Target and Distribution Zones make up the data SMP/E requires to install and track the system's software. It contain a great deal of information. All this information can be displayed using:

- The LIST command creating a hard copy which lists information about the system
- REPORT command which checks, compares, and also generates a hardcopy of the information about zone content
- QUERY Dialogues via ISPF
- SMP/E CSI API which can be used to write application programs to query the content of the system.

2.9.1 SMP/E basic commands

RECEIVE

The RECEIVE process is used to take a SYSMOD which is outside of SMP/E and stage them into the SMP/E Library domain which begins to construct the CSI entries that

describe them. This allows them to be queried for input into later processes. More recently the source can be electronic from a web site, although usually it comes from a tape or even a third party vendor media.

This process's role accomplishes several tasks:

- Constructs entries in the Global Zone for describing the SYSMOD
- Ensuring the SYSMOD is valid such as the syntax for modification control statements (MCS) associated to the products installed in the CSI
- Installing the SYSMOD into the libraries. As an example the PTF temporary store library.
- Assessing the HOLDDATA to ensure errors are not introduced.

During the RECEIVE processing the MCS for each SYSMOD is copied to an SMP/E temporary storage area called the SMPPTS data set containing the inline element replacement or update for that SYSMOD.

Note: There are also RELFILEs which package the elements in relative files that are separate from MCSs which are mostly used by function SYSMODs. Relative files are stored in another temporary storage area called SMPTLIB data sets.

SMP/E updates the global zone with information about the SYSMODs that is has received:

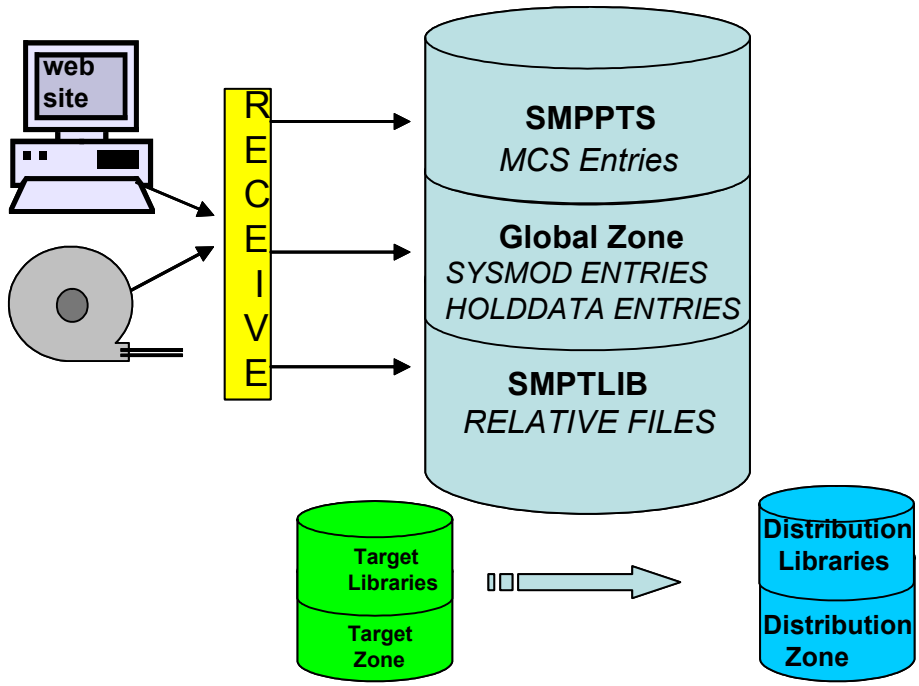


Figure 2-1

In the course of maintaining the system, you need to install service and process the related HOLDDATA. Assume IBM has supplied you with a service tape (such as a CBPDO or ESO tape) and you want to install it on the system. The first step is to receive the SYSMODs and HOLDDATA that are contained on the tape by:

```
SET BDY(GLOBAL)
RECEIVE
```

Use of these commands, SMP/E receives all the SYSMODs and HOLDDATA on the tape.

Note: To RECEIVE only HOLDDATA that may require special handling or that are in error you can use this command. It is important that you receive HOLDDATA into SMP/Es repository as soon as possible.

```
SET BDY(GLOBAL)
RECEIVE HOLDDATA
```

In order to RECEIVE only SYSMODs for installation into the Global Zone:

```
SET BDY(GLOBAL)
RECEIVE SYSMODS
```

For SYSMODs for a specific product (for example, WebSphere Application Server for z/OS) including HOLDDATA:

```
SET BDY(GLOBAL)
RECEIVE FORFMID(H28W500)
```

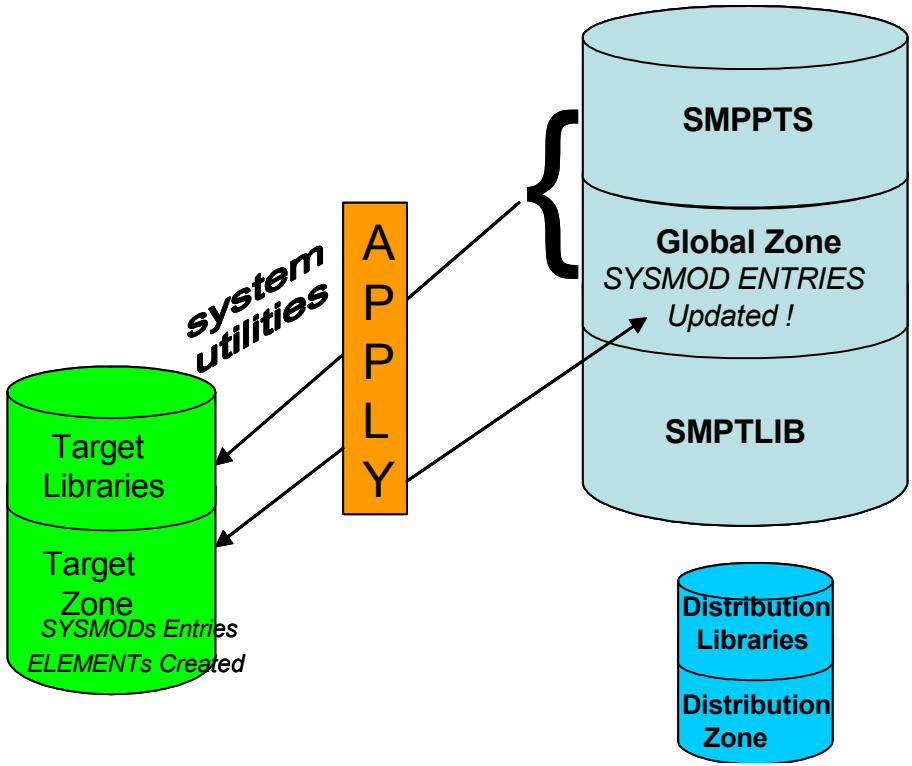
APPLY

The APPLY process instructs SMP/E which of the received SYSMODs are to be selected for installation in the target libraries (TLIBs). SMP/E also ensures that all other required SYSMODs (prerequisites) have been installed or are being installed concurrently as well as in the proper sequence. The source of the elements is the SMPTLIB data sets, the SMPPTS data set or indirect libraries depending on how is was packaged. This phase of the SMP/E process entails:

- Executing the appropriate utility to install the SYSMOD into the target library depending on the type of input text supplied and target module being changed.
- Ensuring the relationship of the new SYSMOD is correct with other SYSMODs in the target zone.
- The CSI is modified displaying the updated modules.

Important: The APPLY command updates the system libraries and should be carefully used on a live Production System. We recommend to initially use a copy of the Production Target Libraries and Zones.

The Target zone reflects the content of the target libraries so therefore, after the utility is completed and zone updated it will accurately reflect the status of those libraries.



The APPLY processing is where the target zone is accurately updated:

- All SYSMOD entries in the Global Zone are updated to reflect that the SYSMOD has been applied to the target zone
- The target zone accurately reflects each SYSMOD entry applied. Element entries (such as MOD and LMOD) are also created in the target zone.
- BACKUP entries are created in the SMPSCDS data set to the SYSMOD can be restored, if at all necessary.

Similar to the RECEIVE process the APPLY command has many different operands for flexibility to select SYSMODs you would like to see for installation in the target libraries and provides an assortment of output. The directives used instruct SMP/E what you want installed.

To install only PTF SYSMODs:

```
SET BDY(ZOSTGT1)
APPLY PTFS.
```

In order to select PTF SYSMODs, you name them in the directives:

```
SET BDY(ZOSTGT1)
APPLY SELECT(UZ00001, UZ00002).
```

In some instances you may only want to install just corrective fixes (APARS) or user modifications (USERMODs) into the target library:

```
SET BDY(ZOSTGT1)
APPLY APARS
    USERMODS.
```

In other instances you may want to update a selected product from a distribution tape:

```
SET BDY(ZOSTGT1).
APPLY PTFS
    FORFMID(H28W500).

or

SET BDY (ZOSTGT1).
APPLY FORFMID(H28W500).
```

Attention: In the two cases above, SMP/E applies all applicable PTFs for the FMID. Unless you specify otherwise, PTFs are the default SYSMOD type.

Using APPLY CHECK

There may be times when you want to see which SYSMODs are included before you actually install them. You can do this with the CHECK operand by entering commands such as the following:

```
SET BDY(MVSTGT1)
APPLY PTFS
APARS
FORFMID(HOP
1)GROUPEXTEND CHECK
```

After these commands are processed, you can check the SYSMOD Status report to see which SYSMODs would have been installed if you had not specified the CHECK operand. If you are satisfied with the results of this trial run, you can enter the commands again, without the CHECK operand, to actually install the SYSMODs.

ACCEPT

Once the SYSMOD has been installed into its Target library and “tested” it then is ACCEPTed. This step is very similar to the APPLY that is taking the selected SYSMODs and installing them into the associated Distribution libraries. You specify operands on the ACCEPT command that tell SMP/E which of the received SYSMODs are to be selected for installation. During this phase SMP/E also ensures that the correct functional level of each element is selected.

Important: There is a “stop” ACCEPT processing that SMP/E provides so you can ensure all prerequisites are satisfied before the installation of the SYSMODs. This is s check for you to see what will happen (assist you in detecting problems) without actually modifying the Distribution libraries.

The ACCEPT performs the follow tasks:

- CSI entries are updated with the targeted elements in the Distribution Zone.
- The targeted elements are either rebuilt or created in the Distribution libraries using the content of the SYSMOD as input depending on the utility
- Similar to the APPLY, it will verify the Target Zone CSI entries for the affected modules and SYSMODs ensuring they are consistent with the library content
- Performs housekeeping of obsolete or expired elements. It will delete the Global Zone CSI entries, PTS members and SMPTLIBs for those SYSMODs affected. As an example Global zone SYSMOD entries and MCS statements in the SMPPTS data set are deleted for those SYSMODs that have been accepted into the Distribution zone.

Note: If you wish that SMP/E does not perform the Global zone clean up, you have the option to indicate this and the information is saved.

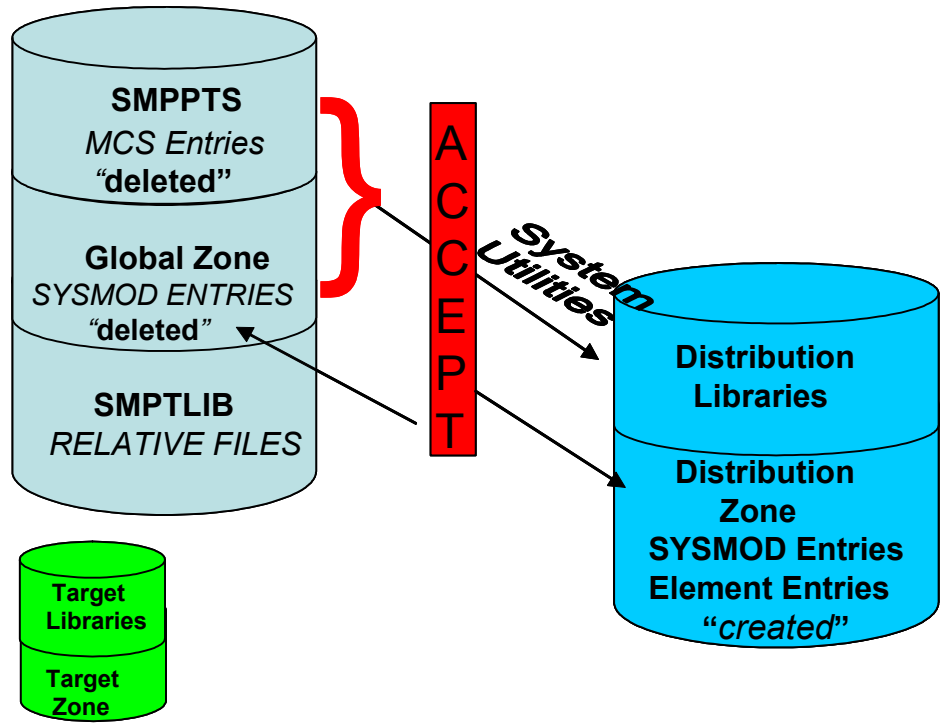


Figure 2-2

Important: Should the SYSMOD be in error, do not ACCEPT it. By using the RESTORE process that takes the modules updated by the selected SYSMOD and rebuilds the copies in the Target libraries from the specific modules in the Distribution libraries as input. Also the RESTORE updates the Target zone CSI entries to reflect the removal of the SYSMOD. When ACCEPT processing is completed, there is **no way** it can be backed out, therefore the changes are permanent.

After applying the SYSMODs into the Target zone, you can then tell SMP/E to install only the eligible PTF SYSMODs into the Distribution zone:

```
SET BDY(ZOSDLB1).
ACCEPT PTFS.
```

To install PTF SYSMODS selecting the particular ones:

```
SET BDY(ZOSDLB1)
ACCEPT SELECT(UZ00001,UZ00002).
```

There are situations where you may want to update a particular product with all SYSMODs:

```
SET BDY(ZOSDLB1)
ACCEPT PTFS
   FORFMID(H28W500).

   or

SET BDY(ZOSDLB1)
ACCEPT FORFMID(H28W500).
```

Note: The the two cases above, SMP/E accepts all applicable PTFs for the product whose FMID is H28W500 located in the Distribution zone ZOSDLB1.

ACCEPTING Prerequisite SYSMODs

When installing a SYSMOD, you may not know if it has prerequisites. SOmetime an ERROR SYSMOD is held). In these situations, you can direct SMP/E to check whether an equivalent (or superseding) SYSMOD is available by specifying the GROUPEXTEND operand:

```
SET BDY(ZOSDLB1).
ACCEPT PTFS
   FORFMID(H28W500)
   GROUPEXTEND.
```

Note: If SMP/E can not find a required SYSMOD, it looks for and uses a SYSMOD that supersedes the required one.

A good way to see which SYSMODs are included before you actually install them is by using the CHECK operand:

```
SET BDY(ZOSTGT1).
ACCEPT PTFS
   FORMFMID(H28W500)
   GROUPEXTEND
   CHECK.
```

ACCEPT Reporting

When this last phase is completed., these reports will assist you to assess the results:

- SYSMOD Status Report - provides a summary of the processing that took place for each SYSMOD, based on the operands you specified on the ACCEPT command.
- Element Summary Report - provides you with a detailed look at each element affected by the ACCEPT processing and which libraries installed

- Causer SYSMOD Summary Report - provides a list of SYSMODs which caused other SYSMODs to fail and descries the errors that must be fixed in order to be successfully processed.
- File Allocation Report - provides a list of the data sets used for the ACCEPT processing and supplies information about these data sets.

Overall SMP/E process flow

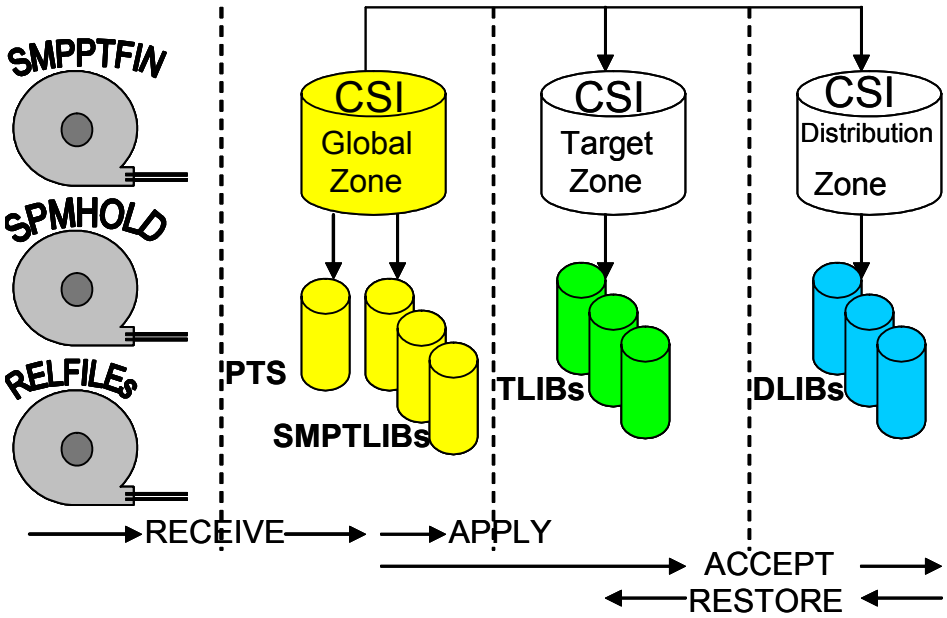


Figure 2-3

Example of CSI VSAM Cluster Definition

```
//DEFINE   JOB   'accounting info',MSGLEVEL=(1,1)
//STEP01   EXEC  PGM=IDCAMS
//CSIVOL   DD    UNIT=3380,VOL=SER=volid1,DISP=SHR
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
  DEFINE CLUSTER(                              -
                 NAME(SMPE.SMPCSI.CSI)         -
                 FREESPACE(10 5)               -
                 KEYS(24 0)                    -
                 RECORDSIZE(24 143)            -
                 SHAREOPTIONS(2 3)             -
        VOLUMES(volid1)              -
                )                              -
            DATA(                              -
                 NAME(SMPE.SMPCSI.CSI.DATA)    -
                 CONTROLINTERVALSIZE(4096)     -
                 CYLINDERS(250 20)             -
                )                              -
           INDEX(                              -
                 NAME(SMPE.SMPCSI.CSI.INDEX)   -
                 CYLINDERS(5 3)                -
                )                              -
          CATALOG(user.catalog)
/*
```

Figure 2-4

SMP batch Job example

```
//SMPJOB   JOB 'accounting info',MSGLEVEL=(1,1)
//SMPSTEP  EXEC SMPPROC
//SMPPTFIN DD  ... points to the file or data set that contains
//*             the SYSMODs to be received
//SMPHOLD  DD  ... points to the file or data set that contains
//*             the HOLDDATA to be received
//SMPTLIB  DD  UNIT=3380,VOL=SER=TLIB01
//SMPCNTL  DD  *
  SET      BDY(GLOBAL)        /* Set to global zone       */.
  RECEIVE  SYSMOD             /* receive SYSMODS and      */
           HOLDDATA           /* HOLDDATA                 */
           SOURCEID(MYPTFS)   /* Assign a source ID       */
                              /*                          */.
  LIST     MCS                /* List the cover letters   */
           SOURCEID(MYPTFS)   /* for the SYSMODs          */
                              /*                          */.
  SET      BDY(TARGET1)       /* Set to target zone       */.
  APPLY    SOURCEID(MYPTFS)   /* Apply the SYSMODs        */
                              /*                          */.
  LIST     LOG                /* List the target zone log */.
/*
```

Figure 2-5

New section starts here:

2.10 When is SMP/E used?

The system programmer responsibilities include base z/OS and IBM software product problem determination and resolution. When a problem is reported, the system programmer must determine whether the problem is the result of local customization or the problem is with IBM code. The systems programmer will call IBM defect support when IBM code is suspect. Typically, defect support will recommend a fix for the system. The systems programmer would use SMP/E to verify the status of this fix on the system. If the fix is absent, then SMP/E would be used to ‘receive’ and ‘apply’ the fix.

To avoid problems, the systems programmer should routinely apply the latest collection of fixes to the base z/OS and other IBM software products. The most aggressive schedule would be monthly, however, it is common for this schedule to be quarterly or semi-annual.

A new or modified business application might expose an IBM code defect. When this occurs, IBM defect support would provide the systems programmer a fix to apply. The fix may require pre-requisite fixes. This can be time consuming and you, the systems programmer, are on a visible critical path. It can bring a significant amount of negative attention. Applying routine maintenance to the system can avoid this situation.

It is in the best interest of the business and the systems programmer to reduce the amount of time required to resolve a code defect which surfaced in your business production environment. A suggested fix may already be applied, therefore, it is immediately known that the problem has not been adequately identified. The best system programmers plan and execute activities which keep them from being on the critical path of processing mission critical business workload, installation of new equipment and business opportunities made possible by exploiting previously unused software feature and functions.

2.11 Why is SMP/E used?

SMP/E provides an excellent inventory, cross-reference of software dependencies and mechanism for applying software maintenance. This insures integrity of the overall system. The mechanics of using SMP/E to maintain a system can be quite simple and easy. How SMP/E works and the knowledge needed to package fixes and software products for SMP/E is significantly complex. IBM provides instructions to follow with all SMP/E packaged fixes, software upgrades and products. The system programmer follows the instructions shipped with the fix, upgrade or product.

2.12 Terminology

As is the case with many technical, scientific and medical disciplines, terminology is the key to describing, understanding and being skilled with a specific discipline. SMP/E falls into this category. We will briefly discuss SMP/E specific terminology, then how a system programmer uses SMP/E. Reference Figure 2-6 on page 2-67 as you read through this terminology section.

2.12.1 Communicating with IBM Defect Support

Defect support will typically recommend a **'PTF'** to fix a problem. The system programmer would use SMP/E to determine absence or presence of the 'PTF' which is

uniquely identified by a seven digit alphanumeric string such as UQ80526. A 'PTF' falls into an SMP/E classification called **'SYSMOD'**. If the problem is identified as a newly reported IBM code defect with no code fix, then IBM opens an **'APAR'**.

- **PTF - P**rogram **T**emporary **F**ix

  When a problem with a software element is discovered, IBM supplies its customers with a tested fix for that problem. This fix comes in the form of a program temporary fix (PTF). Although you may not have experienced the problem the PTF is intended to prevent, it is wise to install the PTF on your system. The PTF SYSMOD is used to install the PTF, thereby preventing the occurrence of that problem on your system. Usually, PTFs are designed to replace or update one or more complete elements of a system function.

- **APAR - A**uthorized **P**rogram **A**nalysis **R**eport

  In critical situations, a code fix or problem bypass will be provided in the form of an APAR fix. The code will then be officially tested and made generally available in the form of a PTF. The APAR fix would, then be replaced by the PTF.

- **SYSMOD - Sys**tem **Mod**ification

  Software, whether it is a product or service, consists of elements such as macros, modules, source, and other types of data (such as CLISTs or sample procedures). For software to be installed by SMP/E, it must include control information (SMP/E instructions) for the elements. The combination of elements and control information is called a system modification, or SYSMOD. **PTFs and APARs are both SYSMODs**

Defect support may inquire about your systems '**PUT level**' before starting, so they have a general idea which PTFs are applied. The system programmer should be able to use SMP/E to determine the PUT level immediately.

- **PUT** level - **P**rogram **U**pdate **T**ape

  Approximately once a month, IBM makes an accumulation of the latest PTFs available on tape. Applying this accumulation of PTFs is called a PUT level update. Generally, PUT levels are in the form of YYMM such as PUT0409, September 2004. However, it is possible that PUT levels will skip a couple months during the year. When this occurs, PUT0409, represents the 9th PUT level during 2004. Defect support will know how to interpret the meaning of the PUT level you provide.

2.12.2 Consolidate Software Inventory (CSI) and zones

SMP/E is a collection of data sets (aka libraries) which are all inter related. This collection of data sets includes the **'target'** libraries which are used to load and run the system. When the system is initially installed, the target libraries are created from a copy of the **'distribution'** libraries. PUT levels and PTFs will are **'APPLIED'** to the target libraries. Once a system programmer is confident the PTFs or PUT level will not need to

be backed out of the system, this software maintenance will be **'ACCEPTED'** into the distribution libraries. The collection of target libraries is given a **'Target Zone'** name and the collection of distribution libraries is given a **'Distribution Zone'** name.

- **Target Zone**

  A name given to the collection of all elements (modules, macros, source, ISPF panels, etc.) used to maintain and run the system. The SMP/E command to update the target zone is **APPLY**.

- **Distribution Zone**

  A name given to a collection of all elements (modules, macros, source, ISPF panels, etc.) used for initial system build, then used to restore elements resulting from a back out of elements applied to the target zone. The SMP/E command to update (permanently commit) changes to the distribution zone is **ACCEPT**.

Before software maintenance can be applied to the target zone, the software maintenance must be '**RECEIVED**' in the **Global Zone.** As you might expect with a name like global, it has a significant amount of information related to the subordinate target and distribution zones, however, it is not directly involved in the operational system.

- **Global Zone**

  A name given to data sets and records within those data sets used for updating and tracking status of previous updates to the target and distribution zones as well as the location used to 'receive' maintenance. The global zone also contains information that (1) enables SMP/E to access target and distribution zones, and (2) enables you to tailor aspects of SMP/E processing.

SMP/E uses **'CSIs'** to keep records of the system. The CSI includes everything we just read about in this terminology section.

- **CSI - Consolidated Software Inventory**

  The CSI keeps track of all elements found in the aggregate of global, target and distribution libraries or zones.

2.12.3 Maintenance

Before or during SMP/E execution to receive maintenance, it is best to apply the **'HOLDDATA'**. The holddata will identify new list of **'PE'** PTFs, changes to pre-requisite dependencies, comments, documentation changes and actions required as a result of applying specific PTFs.

- **PE - P**TF in **E**rror

  Any code or elements applied via PTF packaging found to have a problem is marked with a PE status. The code and elements associated with a PE PTF will not be applied.

- **HOLDDATA**

By receiving the HOLDDATA as soon as possible, you make sure SMP/E has the most current information available. Therefore, if you try to install any PTF in response to a problem on your system and that PTF is in error, SMP/E knows this and warns you so you can weigh the effect of installing the known problem against the effect of fixing the problem you have encountered. Several different HOLD reasons exists:

- Performing some action before installing the SYSMOD (USER HOLD)
- The PTF is in error and awaiting correction (ERROR HOLD)
- The system needs to perform an action before it is installed (SYSTEM HOLD)

Once the maintenance is successfully received and all USER and SYSTEM HOLDs are addressed, then the elements embedded in this PTF or PUT level maintenance is applied to the target libraries. SMP/E JCL does not explicitly have DD statements for each target library because it is not practical. SMP/E makes use of **DDDEFs** located in the CSI instead of having JCL DD statement coded for each and every target library.

- **DDDEF** - **DD**name **Def**inition

  The SMP/E instructions include a ddname as the storage location for new elements or updated elements. SMP/E maintains a list of DDDEF entries which include the ddname with associated data set name or HFS file. This DDDEF entry is used to resolve the ddname allocation request.

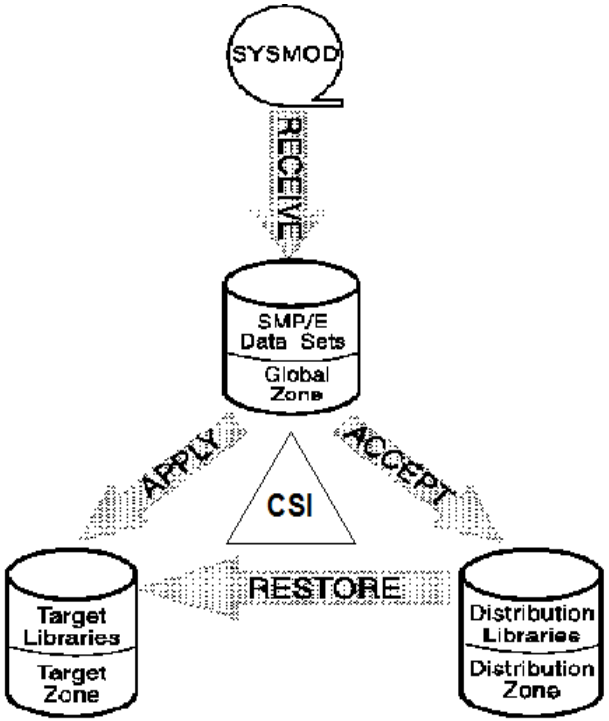


Figure 2-6 Flow of SMP SYSMOD Processing

SMP/E '**Service'** is a term inclusive of APARs, PTFs, PUT Levels, PE PTFs and HOLDDATA. Service can be obtained electronically via the intranet. SMP/E **'CBPDO'** is a full product update. IBM also packages a complete operating system with selected products pre-installed for installation without the need to use SMP/E. This is called a ServerPac.

2.13 Local SMP/E Environments

2.13.1 Production, Test and Emergency Fix

It is common and recommended that fixes never be applied directly to a production system. A typical SMP/E environment may consists of three mutually exclusive copies of the SMP/E libraries that contain the software used by the z/OS system. All three copies will be identical initially. The three SMP/E environments are needed for **1)** the **production system**, **2)** a **test system** containing the next **scheduled update** of all newly identified fixes, and **3)** a **mirror** copy **of** the **production** SMP/E libraries to receive,

apply and **test** an **emergency fix** before introducing the fix into the production environment.

2.13.2 Base z/OS and Product CSIs

It is common and recommended to define an SMP/E CSI environment for the base z/OS and separate SMP/E CSI environments for each product. The base z/OS and optional products typically are shipped with instructions and JCL to define the SMP/E CSI for each individually. Creating a separate SMP/E CSI for a new product install is a matter of following instructions and submitting JCL provided with the product.

2.14 How to use SMP/E

SMP/E can be run either using batch jobs or using dialogues under Interactive System Productivity Facility (ISPF). SMP/E dialogues help you interactively query the SMP/E database, as well as create and submit jobs to process SMP/E commands. Batch facilities are best for receiving, applying and accepting maintenance while ISPF facilities are best to use while working with IBM defect support to answer questions about software levels and presence or absence of specific PTFs. We will first discuss submitting SMP/E JCL and commands in batch, then SMP/E from ISPF.

2.14.1 SYSMOD

A SYSMOD is the actual package containing instructions SMP/E needs to install and track system modifications. SYSMOD are composed of two parts:

- Modification Control Statement (MCS) designated by ++ as the first two characters, that tells SMP/E:
  - What elements are being updated or replaced
  - How the SYSMOD relates to product software and other SYSMODs
  - Other specific installation information

Maintenance and products can be installed without coding any MCS statements. The MCS statements are created by whoever packaged the maintenance or product. SMP/E ‘commands’ are used to tell the SMP/E program where to locate and read these MCS instructions packaged with the system elements. You may frequently see SYSMOD and PTF used interchangeably, however, there is a difference. PTF is the most common type of SYSMOD.

2.14.2 JCL

The SMP/E JCL and commands are very simple as a result of all the target, distribution and work data sets allocation using DDDEF entries. The JCL below could be used to receive sysmods and holddata into the MVS global zone.

```
//SMPJCL   EXEC PGM=GIMSMP,
//        PARM='CSI=MVS.GLOBAL.CSI'
//SMPPTFIN DD  DSN=
//SMPHOLD  DD  DSN=
//SMPLOG   DD  SYSOUT=*
//SMPCNTL  DD  *
   SET BOUNDARY(GLOBAL).
   RECEIVE.
```

The SMP/E program, GIMSMP, is passed the name of the data set which is the global CSI, MVS.GLOBAL.CSI in this case. The SMP/E program needs several input and output ddnames. SMP/E will expect to read instructions and sysmods from the data set name associated with SMPPTFIN ddname and it will expect to read holddata from data set name associated with SMPHOLD ddname. SMP/E will write its message to SMPLOG ddname. SMPCNTL provides the SMP/E its commands. The SET command tells SMP/E which zone that the following commands are directed. In this case SMP/E is told the following commands are intended for the global zone.

2.14.3 RECEIVE Command

The RECEIVE command directed to the global zone will **1)** read sysmods with instructions, **2)** read holddata, **3)** write holddata and sysmods with instructions to SMPPTS DDDEF data set **4)** write SMP/E messages and **5)** update the global data set with information about what it received. When completed, the global data set will show date, time and status for each sysmod successfully received.

2.14.4 APPLY Command

The only input and output hardcoded JCL ddnames used to APPLY sysmods is the SMPCNTL which provides SMP/E program its commands and ddname to write messages. The SET BOUNDRY command is directing following commands to MVST target zone. The target zone name is arbitrarily chosen by the system programmer.

```
//APPLY   EXEC PGM=GIMSMP,
//       PARM='CSI=MVS.GLOBAL.CSI'
//SYSPRINT DD  SYSOUT=*
//SMPLOG   DD  SYSOUT=*
//SMPCNTL  DD  *
  SET BOUNDARY(MVST).
  APPLY SELECT(UW85091,UW85613)
      GROUPEXTEND COMPRESS(ALL)
```

```
BYPASS(HOLDSYSTEM) CHECK.
```

The above APPLY command will 1) take ‘selected’ sysmods from the global zone SMPPTS data set, 2) read the instructions and elements packaged in the sysmods and 3) execute the instructions which will result in adding, replacing and modifying elements in the target libraries. Several operands are present on the APPLY command. Specific PTF sysmods are SELECTed to be applied. The GROUPEXTEND operand tells the program to include any required prerequisite or corequisite sysmods that have been received but not specifically selected. COMPRESS(ALL) tells the program to regain unused space in a target library when needed. BYPASS(HOLDSYSTEM) tells the program that the system programmer has taken all specified system actions included as comments on the specific sysmods in SYSTEM HOLD status. CHECK tells the program this is a trial run and please produce messages that would indicate probably of failure or success once the CHECK is removed.

ISPF

2.14.5 ACCEPT Command

Once the system programmer is confident the applied sysmods will not be backed out, the same JCL used to APPLY is used to permanently ACCEPT the changes. Again, in this case ACCEPT commands is directed to the distribution zone with a given name, MVSD.

```
SET BOUNDARY(MVSD)
ACCEPT SELECT(UW85091,UW85613).
```

2.14.6 LIST Command

The same JCL used for APPLY processing can be used. The LIST command can be directed to the global, target or distribution zone to produce a listing of all known PTFs, selective PTFs along with their status (receive, applied, superseded and accepted) or DDDEFs, etc. An example to list a single PTF (UW85091), list all PTFs and list all DDDEF entries in the target zone (MVST) follows.

```
SET BOUNDARY(MVST).
LIST SYSMODS(UW85091).
LIST SYSMODS.
LIST DDDEF.
```

2.15 ISPF

Using ISPF SMP/E Panels is the best method to use when working with defect support....to answer questions about existing maintenance levels and to determine whether or not a PTF or SYSMOD is applied.

Figure 2-7 on page 2-71 is an example of the SMP/E Primary Option Menu. Tutorial and PF1 help panels exist in this ISPF application.

```
---------------------- SMP/E PRIMARY OPTION MENU ---------------- SMP/E 32.06
===> 3


    0  SETTINGS           - Configure settings for the SMP/E dialogs
    1  ADMINISTRATION     - Administer the SMPCSI contents
    2  SYSMOD MANAGEMENT  - Receive SYSMODs and HOLDDATA
                            and install SYSMODs
    3  QUERY              - Display SMPCSI information
    4  COMMAND GENERATION - Generate SMP/E commands
    5  RECEIVE            - Receive SYSMODs, HOLDDATA and
                            support information
    6  MIGRATION ASSISTANT- Generate Planning and Migration Reports

    D  DESCRIBE           - An overview of the dialogs
    T  TUTORIAL           - Details on using the dialogs
    W  WHAT IS NEW        - What is New in SMP/E

Specify the name of the CSI that contains the global zone:
   SMPCSI DATA SET   ===> 'MVS.GLOBAL.CSI'
(Leave blank for a list of SMPCSI data set names.)

Specify YES to have DD statements for SYSOUT and temporary
data sets generated. Specify NO, to use DDDEFs.
   Generate DD statements   ===> NO
```

Figure 2-7 SMP/E Primary Option Menu

IBM PTF packaging instructions typically associate a PTF number with a PUT level. The PUT levels are given ‘SOURCEIDs’ names such as PUT0403. Therefore, if asked what PUT is the latest PUT level for the base z/OS or an option software product, a ‘SOURCEID’ query will help provide that information. Figure 2-8 on page 2-71 selects sourceid and Figure 2-9 on page 2-72 requests a zone to be selected for the sourceid query. When working with IBM defect support, you want to always select the target zone.

```
                              QUERY SELECTION MENU
===> 3

     1  CSI QUERY           - Display SMPCSI entries
     2  CROSS-ZONE QUERY    - Display status of an entry in
                              all zones
     3  SOURCEID QUERY      - Display SOURCEIDs for specified zone

     D  DESCRIBE            - Overview of using QUERY

     T  TUTORIAL            - Information on using QUERY

To return to the SMP/E primary option menu, enter  END .
```

Figure 2-8 SourceID Query Selection - Location of PUT level information

```
                                        QUERY - ZONE SELECTION
===>

Select the zone that you want to query:

S     NAME      TYPE       CSI DATA SET
      CSQ531D   DLIB       CSQ531.CSQ531D.CSI
      CSQ531T   TARGET     CSQ531.CSQ531T.CSI
      GLOBAL    GLOBAL     MVS.GLOBAL.CSI
      MVSD      DLIB       MVS.MVSD.CSI
s     MVST      TARGET     MVS.MVST.CSI
****************************** Bottom of data ***
```

Figure 2-9 Query PUT levels within target zone

All SOURCEID PUT levels will be displayed among any other SOURCEIDs used by you are within SMP/E packaging. Selecting a PUT level, Figure 2-10 on page 2-72, will display all the PTFs included in the PUT level. In this example, approximately 20 PTFs are in the March 2004 PUT level. The latest SOURCEID PUT levels are may not include all PTFs released for that PUT level. Applying an emergency fix may result in the PUT level SOURCEID addition in the CSI because it is the PUT level associated with the PTF. The PUT level SOURCEID would be fully populated with all PTFs in the PUT level when routine maintenance is completed.

If any one of the PTFs from SOURCEID PUT0403, Figure 2-11 on page 2-73, then details about the individual PTF will be displayed, such as status with date and time applied, Figure 2-12 on page 2-73.

```
                        SOURCEID QUERY - SOURCEID SELECTION
===>

Select one SOURCEID for TARGET zone MVST

S     SOURCEID  ACTION
      PUT0303
      PUT0304
      PUT0305
      PUT0306
      PUT0307
      PUT0308
      PUT0309
      PUT0310
      PUT0311
      PUT0312
      PUT0401
      PUT0402
s     PUT0403
      PUT9402
      PUT9404
      PUT9405
```

Figure 2-10 PUT Levels - Select PUT0403 Level - for included PTFs

```
                    SOURCEID PUT0403 QUERY - SYSMOD ENTRY SELEC
===>

Select one SYSMOD to query from TARGET zone MVST

S     NAME      ACTION
      UA09538
      UA09612
      UA09629
      UA09650
      UA09661
      UA09669
      UA09688
      UA09689
      UA09703
      UA09742
      UA09754
      UA09758
      UA09778
      UA09794
      UA09867
      UA10039
      UQ86388
      UQ86511
      UQ86879
      UQ87074
s     UR54489
```

Figure 2-11 List of PTFs within PUT level - Select a specific PTF from the PUT level

```
                          CSI QUERY - SYSMOD ENTRY          Row 1 to :
===>                                                         SCROLL ==

 To return to the previous panel, enter END .


 Primary Command: FIND

 Entry Type:  SYSMOD                                Zone Name: MVST
 Entry Name:  UR54489                               Zone Type: TARGET
 Description:

   Type:    PTF                   Status: APP
   FMID:    HBCNC00
   Date/Time: 04.111   12:20:43   APP


         -------- -------- -------- -------- -------- -------- --------
SUP      AR39670  AR39742  AR40054  AR40055  AR40056  AR40057  AR40914
         AR41057  AR41058  AR41059  AR41061  AR41878  AR42098  AR42099
         AR42100  AR42101  AR42102  AR42103  AR42104  AR42105  AR42106
         AR42107  AR42108  AR44470  AR44497  AR44498  AR45284  AR45285
```

Figure 2-12 SYSMOD(PTF) status info within PUT Level

A SYSMOD (PTF) can be queried directly for the SMP/E Primary Option Menu. After ‘Query’ is selected from the Primary Menu, then the ‘zone’ (MVST), the type of entry (SYSMOD) and specific SYSMOD(PTF) number can be entered to display the details. Figure 2-13 on page 2-74

```
                                    CSI QUERY
===>

Specify the zone, entry type, and name to be queried:

   ZONE NAME    ===> MVST      Name of the zone to be queried.
                               To display a list of all zones,
                               leave blank

   ENTRY TYPE   ===> SYSMOD    Entry type to be queried.
                               To display a list of all valid
                               entry types, leave ENTRY TYPE
                               and ENTRY NAME blank

   ENTRY NAME   ===> UR54489   Entry name to be queried.
                               To display a list of all entries
                               for the selected zone and entry
                               type, leave blank
```

Figure 2-13 Query SYSMOD(PTF) directly w/o going through SourceID panels

2.16 Order and download IBM fixes

While collections of PTFs are typically received from Product Update Tape, PUT, and entire product updates are typically received from tape, CBPDO, emergency fixes can be ordered and download over the Internet. The order and download can typically be completed in 15 minutes.

https://techsupport.services.ibm.com/server/login is an IBM web site, used by system programmers. Any system programmer can register to order and download fixes. The download includes instructions. This web site is used to order z/OS, AIX, and OS/400 fixes. The web site for all IBM servers.

2.17 Summary

A critical responsibility of the system programmer is to work with IBM defect support when a problem surfaces in z/OS or option IBM products. Problem resolution will require the system programmer to receive and apply fixes to the enterprise system.

The SMP/E JCL and commands will be used frequently by a large enterprise z/OS system programmer, however, SMP/E MCS instructions will rarely be coded by the same

system programmer. The product and SYSMOD packaging will include the necessary MCS statements.

3

Hardware systems and LPARs

Objective: As a new z/OS system programmer, you will need to develop a thorough understanding of the hardware that runs the z/OS operating system. z/OS is designed to make full use of mainframe hardware and its many sophisticated peripheral devices.

After completing this topic, you will be able to:

- Discuss S/360 and zSeries hardware design.
- Explain processing units and disk hardware.
- Explain how mainframes differ from PC systems in data encoding.
- List some typical hardware configurations.

3.1 Overview of mainframe hardware systems

This topic provides an overview of mainframe hardware systems, with most of the emphasis on the processor “box.”

Related Reading: For detailed descriptions of the major facilities of z/Architecture, the book *z/Architecture Principles of Operation* is the standard reference. You can find this and other IBM publications at the z/OS Internet Library Web site:

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

Let’s begin this topic with a look at the terminology associated with mainframe hardware. Being aware of various meanings of the terms *systems*, *processors*, *CP*s, and so forth is important for your understanding of mainframe computers.

In the early S/360 days a system had a single processor, which was also known as the central processing unit (CPU). The terms *system*, *processor*, and *CPU* were used interchangeably. However, these terms became confusing when systems became available with more than one processor. This is illustrated in Figure 3-1.

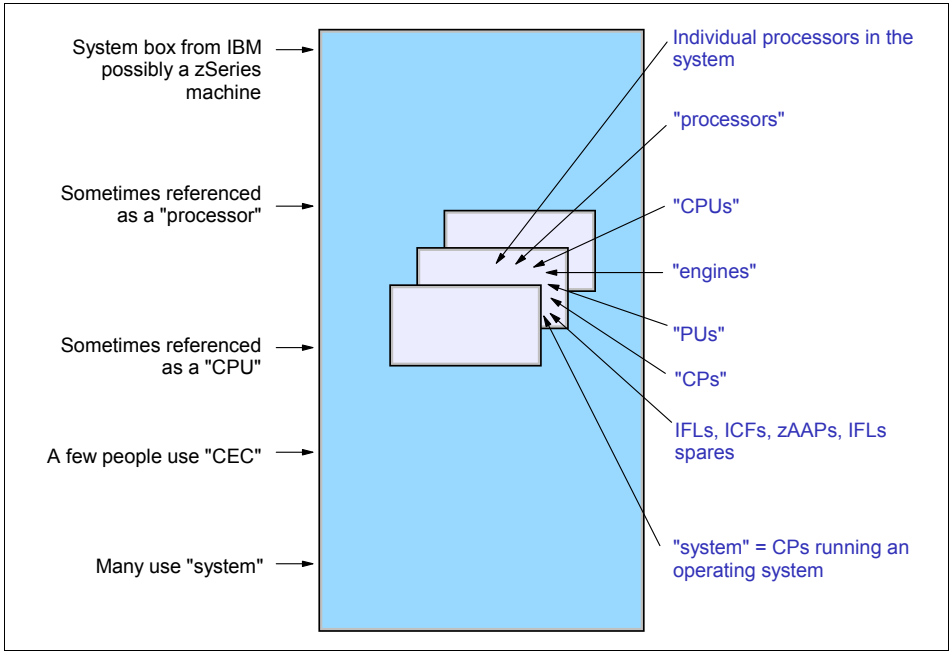


Figure 3-1 Terminology overlap

Processor and CPU can refer to either the complete system box, or to one of the processors (CPUs) within the system box. Although the meaning may be clear from the context of a discussion, even mainframe professionals must clarify which *processor* or

CPU meaning they are using in a discussion. IBM introduced the term CEC (central electronic complex) to unambiguously refer to the "box." However, this term is not widely used.

Partitioning and some of the terms in Figure 3-1 are discussed later in this topic. Briefly, all the S/390 or z/Architecture processors within a CEC are *processing units* (PUs). When IBM delivers the CEC, the PUs are characterized as CPs (for normal work), Integrated Facility for Linux (IFL), Integrated Coupling Facility (ICF) for Parallel Sysplex configurations, and so forth.

In this text, we hope the meanings of *system* and *processor* are clear from the context. We normally use *system* to indicate the hardware box, a complete hardware environment (with I/O devices), or an operating environment (with software), depending on the context. We normally use *processor* to mean a single processor (CP) within the CEC.

3.2 Early system design

The central processor box contains the processors, memory,[1] control circuits, and interfaces for *channels*. A channel provides an independent data and control path between I/O devices and memory. Early systems had up to 16 channels; the largest mainframe machines at the time of writing can have over 1000 channels.

Channels connect to *control units*. A control unit contains logic to work with a particular type of I/O device. A control unit for a printer would have much different internal circuitry and logic than a control unit for a tape drive, for example. Some control units can have multiple channel connections providing *multiple paths* to the control unit and its devices.

Control units connect to *devices*, such as disk drives, tape drives, communication interfaces, and so forth. The division of circuitry and logic between a control unit and its devices is not defined, but it is usually more economical to place most of the circuitry in the control unit.

Figure 3-2 presents a conceptual diagram of a S/360 system. Current systems are not connected as shown in Figure 3-2. However, this figure helps explain the background terminology that permeates mainframe discussions.

[1] Some S/360s had separate boxes for memory. However, this is a conceptual discussion and we ignore such details.

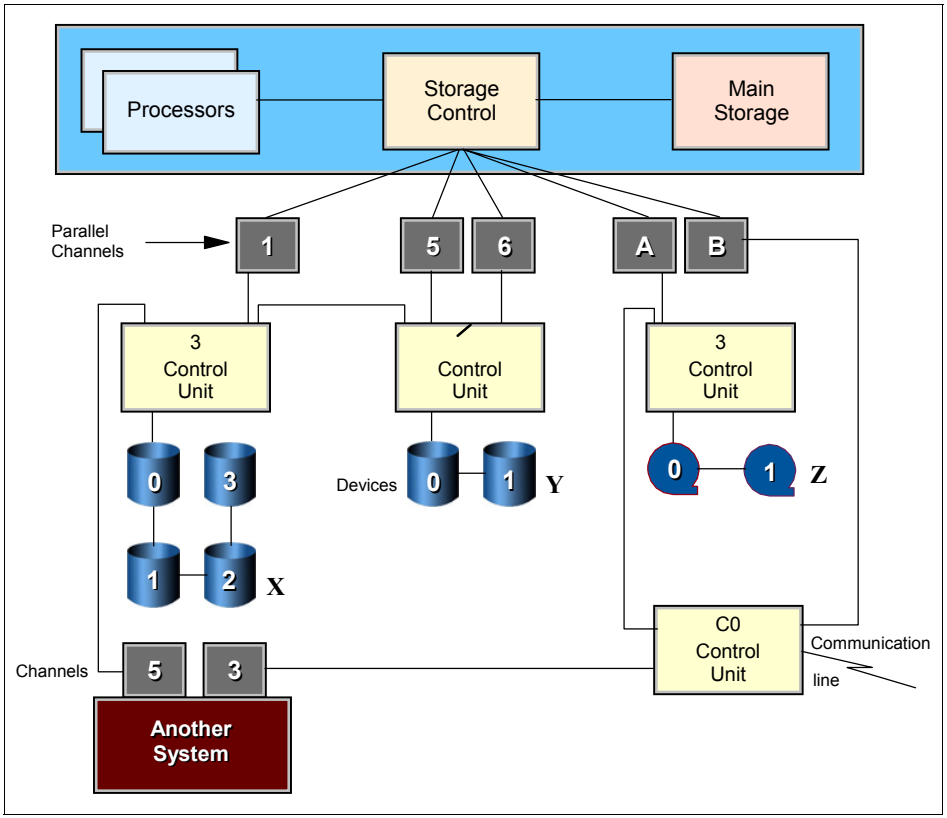


Figure 3-2 Conceptual S/360

The channels in Figure 3-2 are *parallel channels* (also known as *bus and tag channels*, named for the two heavy copper cables they use). A parallel channel can be connected to a maximum of eight control units. Most control units can be connected to multiple devices; the maximum depends on the particular control unit, but 16 is a typical number.

Each channel, control unit, and device has an address, expressed as a hexadecimal number. The disk drive marked with an X in Figure 3-2 has address 132, derived as shown in Figure 3-3.

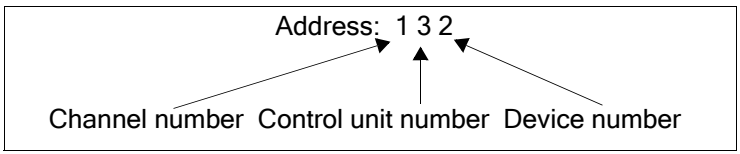


Figure 3-3 Device address

The disk drive marked with a Y in the figure can be addressed as 171, 571, or 671 because it is connected through three channels. By convention the device is known by its lowest address (171), but all three addresses could be used by the operating system to access the disk drive. Multiple paths to a device are useful for performance and for availability. When an application wants to access disk 171, the operating system will first try channel 1. If it is busy (or not available), it will try channel 5, and so forth.

Figure 3-2 contains another S/360 system with two channels connected to control units used by the first system. This sharing of I/O devices is common in all mainframe installations. Tape drive Z is address A31 for the first system, but is address 331 for the second system. Sharing devices, especially disk drives, is not a simple topic and there are hardware and software techniques used by the operating system to control exposures such as updating the same disk data at the same time from two independent systems.

As mentioned, current mainframes are not used exactly as shown in Figure 3-2 on page 80. Differences include:

- Parallel channels are not available on the newest mainframes and are slowly being displaced on older systems.
- Parallel channels have been replaced with ESCON (Enterprise Systems CONnection) and FICON® (FIber CONnection) channels. These channels connect to only one control unit or, more likely, are connected to a *director* (switch) and are optical fibers.
- Current mainframes have more than 16 channels and use two hexadecimal digits as the channel portion of an address.
- Channels are generally known as CHPIDs (Channel Path Identifiers) or PCHIDs (Physical Channel IDs) on later systems, although the term *channel* is also correct. The channels are all integrated in the main processor box.

The device address seen by software is more correctly known as a device number (although the term *address* is still widely used) and is indirectly related to the control unit and device addresses.

3.3 Current design

Current CEC designs are considerably more complex than the early S/360 design. This complexity includes many areas:

- I/O connectivity and configuration
- I/O operation
- Partitioning of the system

3.3.1 I/O connectivity

Figure 3-4 on page 83 illustrates a recent configuration. (A real system would have more channels and I/O devices, but this figure illustrates key concepts.) Partitions, ESCON channels, and FICON channels are described later.

Briefly, partitions create separate logical machines in the CEC. ESCON and FICON channels are logically similar to parallel channels but they use fiber connections and operate much faster. A modern system might have 100-200 channels or CHPIDs.[2] Key concepts partly illustrated here include the following:

- ESCON and FICON channels connect to only one device or one port on a switch.
- Most modern mainframes use switches between the channels and the control units. The switches may be connected to several systems, sharing the control units and some or all of its I/O devices across all the systems.
- CHPID addresses are two hexadecimal digits.
- Multiple partitions can sometimes share CHPIDs. Whether this is possible depends on the nature of the control units used through the CHPIDs. In general, CHPIDs used for disks can be shared.
- An I/O subsystem layer exists between the operating systems in partitions (or in the basic machine if partitions are not used) and the CHPIDs.

An ESCON director or FICON switch is a sophisticated device that can sustain high data rates through many connections. (A large director might have 200 connections, for example, and all of these can be passing data at the same time.) The director or switch must keep track of which CHPID (and partition) initiated which I/O operation so that data and status information is returned to the right place. Multiple I/O requests, from multiple CHPIDs attached to multiple partitions on multiple systems, can be in progress through a single control unit.

The I/O control layer uses a control file known as an IOCDS (I/O Control Data Set) that translates physical I/O addresses (composed of CHPID numbers, switch port numbers, control unit addresses, and unit addresses) into *device numbers* that are used by the operating system software to access devices. This is loaded into the Hardware Save Area (HSA) at power-on and can be modified dynamically. A device number looks like the addresses we described for early S/360 machines except that it can contain three or four hexadecimal digits.

[2] The more recent mainframe machines can have more than 256 channels, but an additional setup is needed for this. The channels are assigned in a way that only two hexadecimal digits are needed for CHPID addresses.

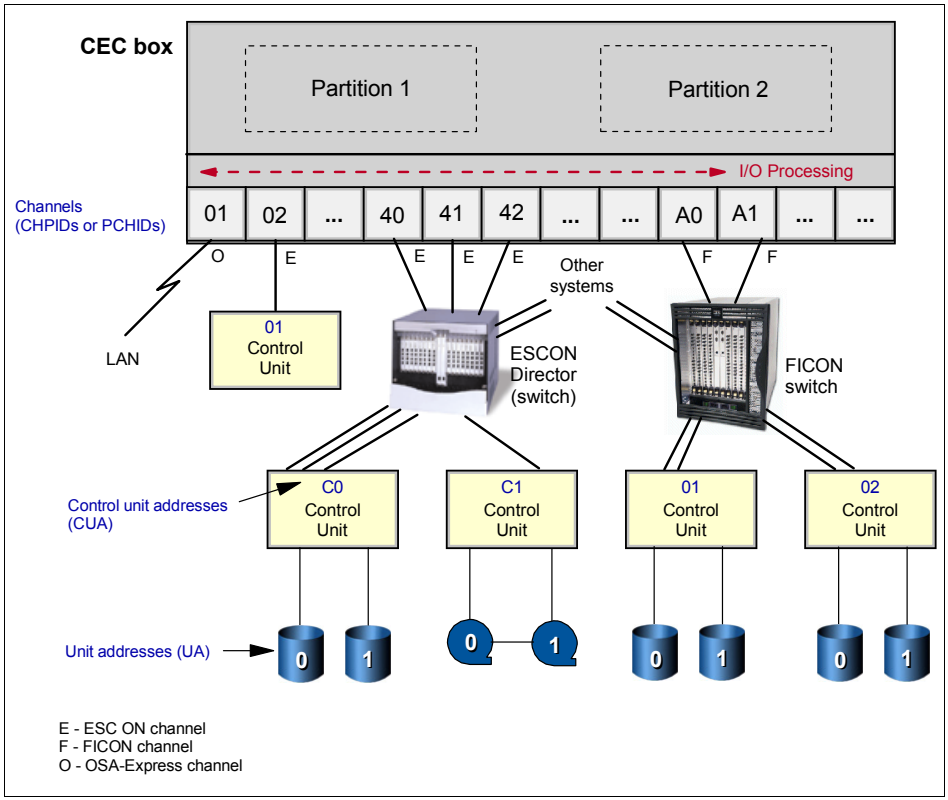


Figure 3-4 Recent system configuration

Many users still refer to these as “addresses” although the device numbers are arbitrary numbers between x'0000' and x’FFFF’. The newest mainframes, at the time of writing, have two layers of I/O address translations between the real I/O elements and the operating system software. The second layer was added to make migration to newer systems easier.

Modern control units, especially for disks, often have multiple channel (or switch) connections and multiple connections to their devices. They can handle multiple data transfers at the same time on the multiple channels. Each device will have a unit control block (UCB) in each z/OS image.

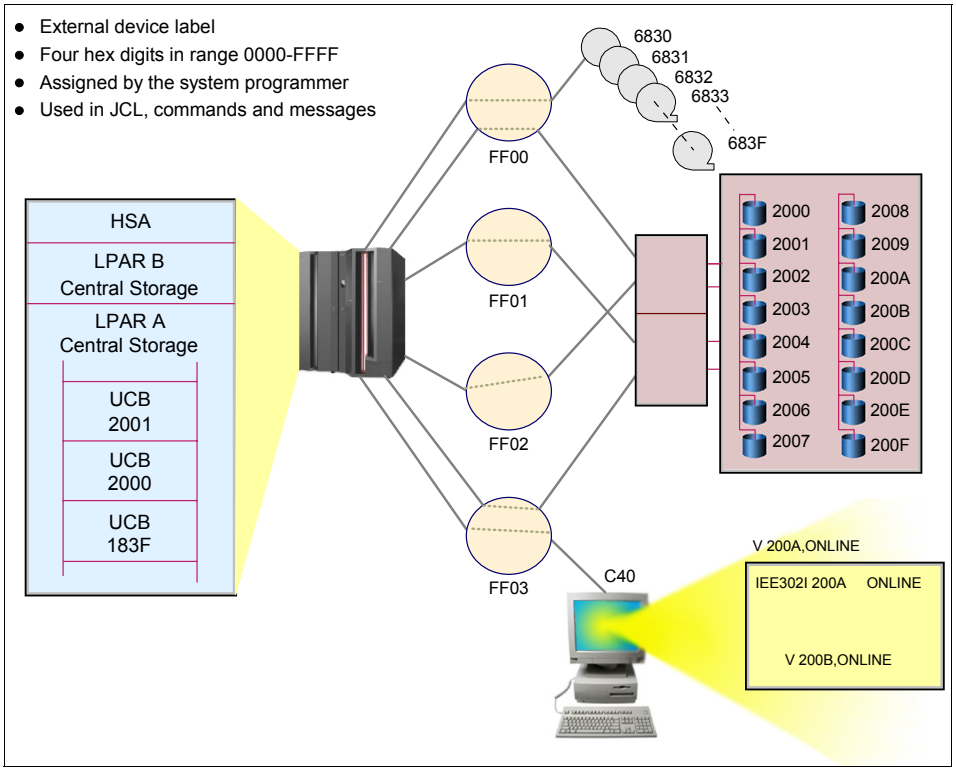


Figure 3-5 Device addressing

3.3.2 System control and partitioning

There are many ways to illustrate a mainframe's internal structure, depending on what we wish to emphasize. Figure 3-6 on page 85, while highly conceptual, shows several of the functions of the internal system controls on current mainframes. The internal controllers are microprocessors but use a much simpler organization and instruction set than zSeries processors. They are usually known as *controllers* to avoid confusion with zSeries *processors*.

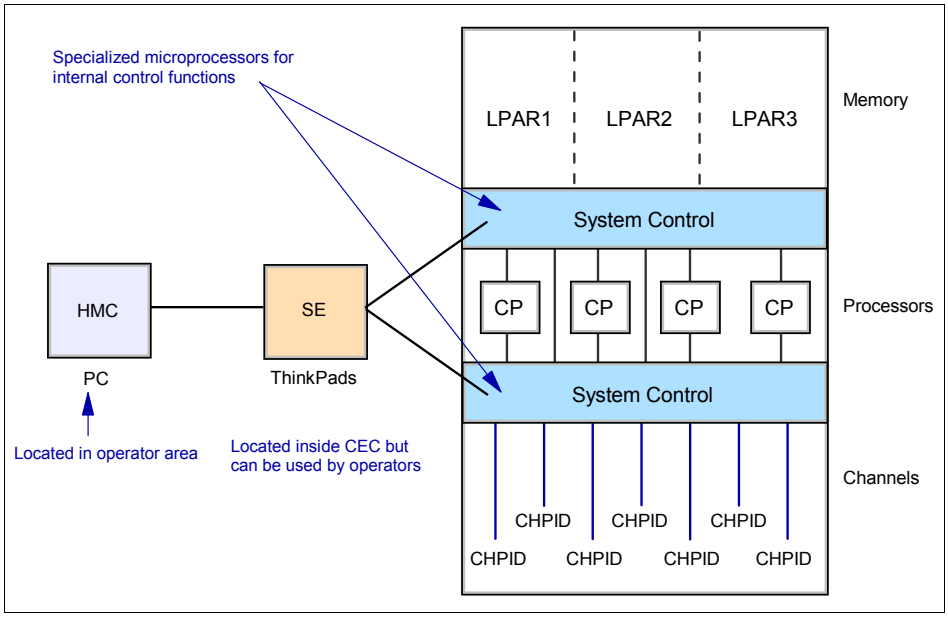


Figure 3-6 System control and partitioning

Among the system control functions is the capability to partition the system into several *logical partitions* (LPARs). An LPAR is a subset of the processor hardware that is defined to support an operating system. An LPAR contains resources (processors, memory, and input/output devices) and operates as an independent system. Multiple logical partitions can exist within a mainframe hardware system.

For many years there was a limit of 15 LPARs in a mainframe; more recent machines have a limit of 30 (and potentially more). Practical limitations of memory size, I/O availability, and available processing power usually limit the number of LPARs to less than these maximums.

Note: The hardware and firmware that provides partitioning is known as PR/SM™ (Processor Resource/System Manager). It is the PR/SM functions that are used to create and run LPARs. This difference between PR/SM (a built-in facility) and LPARs (the result of using PR/SM) is often ignored and the term LPAR is used collectively for the facility and its results.

System administrators assign portions of memory to each LPAR; memory cannot be shared among LPARs. The administrators can assign processors (noted as CPs in Figure 3-6) to specific LPARs or they can allow the system controllers to dispatch any or all the processors to all the LPARs using an internal load-balancing algorithm. Channels

(CHPIDs) can be assigned to specific LPARs or can be shared by multiple LPARs, depending on the nature of the devices on each channel.

A system with a single processor (CP processor) can have multiple LPARs. PR/SM has an internal dispatcher that can allocate a portion of the processor to each LPAR, much as an operating system dispatcher allocates a portion of its processor time to each process, thread, or task.

Partitioning control specifications are partly contained in the IOCDS and are partly contained in a system *profile*. The IOCDS and profile both reside in the *Support Element* (SE) which is simply a ThinkPad (IBM laptop) inside the system. The SE can be connected to one or more Hardware Management Consoles (HMCs), which are desktop personal computers. An HMC is more convenient to use than an SE and can control several different mainframes.

Working from an HMC (or from an SE, in unusual circumstances), an operator prepares a mainframe for use by selecting and loading a profile and an IOCDS. These create LPARs and configure the channels with device numbers, LPAR assignments, multiple path information, and so forth. This is known as a Power-on Reset (POR). By loading a different profile and IOCDS, the operator can completely change the number and nature of LPARs and the appearance of the I/O configuration. However, doing this is usually disruptive to any running operating systems and applications and is therefore seldom done without advance planning.

3.3.3 Characteristics of LPARs

LPARs are, in practice, equivalent to separate mainframes. Each LPAR runs its own operating system. This can be any mainframe operating system; there is no need to run z/OS, for example, in each LPAR. The installation planners may elect to share I/O devices across several LPARs, but this is a local decision.

The system administrator can assign one or more system processors for the exclusive use of an LPAR. Alternately, the administrator can allow all processors to be used on some or all LPARs. Here, the system control functions (often known as microcode or firmware) provide a dispatcher to share the processors among the selected LPARs. The administrator can specify a maximum number of concurrent processors executing in each LPAR. The administrator can also provide weightings for different LPARs; for example, specifying that LPAR1 should receive twice as much processor time as LPAR2.

The operating system in each LPAR is IPLed separately, has its own copy[3] of its operating system, has its own operator console (if needed), and so forth. If the system in one LPAR crashes, there is no effect on the other LPARs.

[3] Most, but not all, of the z/OS system libraries can be shared.

In Figure 3-6 on page 85, for example, we might have a production z/OS in LPAR1, a test version of z/OS in LPAR2, and Linux for S/390 in LPAR3. If our total system has 8 GB of memory, we might have assigned 4 GB to LPAR1, 1 GB to LPAR2, 1 GB to LPAR3, and have kept 2 GB in reserve for some reason. The operating system consoles for the two z/OS LPARs might be in completely different locations.[4]

For most practical purposes there is no difference between, for example, three separate mainframes running z/OS (and sharing most of their I/O configuration) and three LPARs on the same mainframe doing the same thing. With minor exceptions z/OS, the operators, and applications cannot detect the difference.

The minor differences include the ability of z/OS (if permitted when the LPARs were defined) to obtain performance and utilization information across the complete mainframe system and to dynamically shift resources (processors and channels) among LPARs to improve performance.

3.3.4 Consolidation of mainframes

There are fewer mainframes in use today than there were 15 or 20 years ago. In some cases, all the applications were moved to other types of systems. However, in most cases the reduced number is due to consolidation. That is, several smaller mainframes have been replaced with a smaller number of larger systems.

There is a compelling reason for consolidation. Mainframe software (from many vendors) can be expensive and typically costs more than the mainframe hardware. It is usually less expensive (and sometimes *much* less expensive) to replace multiple software licenses (for smaller machines) with one or two licenses (for larger machines). Software license costs are often linked to the power of the system but the pricing curves favor a small number of large machines.

Software license costs for mainframes have become a dominant factor in the growth and direction of the mainframe industry. There are several nonlinear factors that make software pricing very difficult. We must remember that mainframe software is not a mass market situation like PC software. The growth of mainframe processing power in recent years has been nonlinear.

The relative power needed to run a traditional mainframe application (a batch job written in COBOL, for example) does not have a linear relation to the power needed for a new application (with a GUI interface, written in C and Java). The consolidation effect has produced very powerful mainframes. These might need 1% of their power to run an application, but the application vendor often sets a price based on the total power of the machine.

[4] Linux does not have an operator console in the sense of the z/OS consoles.

This results in the odd situation where customers want the latest mainframe (to obtain new functions or to reduce maintenance costs associated with older machines) but they want the *slowest* mainframe that will run their applications (to reduce software costs based on total system processor power).

3.4 Processing units

Figure 3-1 on page 78 lists several different types of processors in a system. These are all z/Architecture processors that can be used for slightly different purposes.[5] Several of these purposes are related to software cost control, while others are more fundamental.

All these start as equivalent processor units[6] (PUs) or engines. A PU is a processor that has not been *characterized* for use. Each of the processors begins as a PU and is characterized by IBM during installation or at a later time. The potential characterizations are:

- Central Processor (CP)

  This is a processor available to normal operating system and application software.

- System Assistance Processor (SAP)

  Every modern mainframe has at least one SAP; larger systems may have several. The SAPs execute internal code[7] to provide the I/O subsystem. An SAP, for example, translates device numbers and real addresses of CHPIDs, control unit addresses, and device numbers. It manages multiple paths to control units and performs error recovery for temporary errors. Operating systems and applications cannot detect SAPs, and SAPs do not use any "normal" memory.

- Integrated Facility for Linux (IFL)

  This is a normal processor with one or two instructions disabled that are used only by z/OS. Linux does not use these instructions and can be executed by an IFL. Linux can be executed by a CP as well. The difference is that an IFL is not counted when specifying the model number[8] of the system. This can make a substantial difference in software costs.

- zAAP

  This is a processor with a number of functions disabled (interrupt handling, some instructions) such that no full operating system can be executed on the processor. However, z/OS can detect the presence of zAAP processors and will use them to

[5] Do not confuse these with the controller microprocessors. The processors discussed in this section are full, standard mainframe processors.

[6] This discussion applies to the current zSeries machines at the time of writing. Earlier systems had fewer processor characterizations, and even earlier systems did not use these techniques.

[7] IBM refers to this as Licensed Internal Code (LIC). It is often known as microcode (which is not technically correct) or as firmware. It is definitely not user code.

[8] Some systems do not have different models; in this case a *capacity model number* is used.

execute Java code (and possibly other similar code in the future). The same Java code can be executed on a standard CP. Again, zAAP engines are not counted when specifying the model number of the system. Like IFLs, they exist only to control software costs.

- Integrated Coupling Facility (ICF)

  These processors run only Licensed Internal Code. They are not visible to normal operating systems or applications. A Coupling Facility is, in effect, a large memory scratch pad used by multiple systems to coordinate work. ICFs must be assigned to LPARs that then become coupling facilities. These topics are discussed in more detail in Chapter 4, "Parallel Sysplex and continuous availability" on page 103.

- Spare

  An uncharacterized PU functions as a "spare." If the system controllers detect a failing CP or SAP, it can be replaced with a spare PU. In most cases this can be done without any system interruption, even for the application running on the failing processor.

- Various forms of Capacity on Demand and similar arrangements exist whereby a customer can enable additional CPs at certain times (for unexpected peak loads, for example).

In addition to these characterizations of processors, some mainframes have models or versions that are configured to operate slower than the potential speed of their CPs. This is widely known as *kneecapping*, although IBM prefers the term *capacity setting,* or something similar. It is done by using microcode to insert null cycles into the processor instruction stream. The purpose, again, is to control software costs by having the minimum mainframe model or version that meets the application requirements. IFLs, SAPs, zAAPs, and ICFs always function at the full speed of the processor since these processors "do not count" in software pricing calculations.[9]

3.5 Multiprocessors

All the earlier discussions and examples assume that more than one processor (CP) is present in a system (and perhaps in an LPAR). It is possible to purchase a current mainframe with a single processor (CP), but this is not a typical system.[10] The term *multiprocessor* means several processors (CP processors) and implies that several processors are used by a copy of z/OS.

All operating systems today, from PCs to mainframes, can work in a multiprocessor environment. However, the degree of integration of the multiple processors varies

[9] This is true for IBM software but may not be true for all software vendors.

[10] All current IBM mainframes also require at least one SAP, so the minimum system has two processors: one CP and one SAP. However, the use of "processor" in the text usually means a CP processor usable for applications. Whenever discussing a processor other than a CP, we always make this clear.

considerably. For example, pending interrupts in a system (or in an LPAR) can be accepted by any processor in the system (or working in the LPAR). Any processor can initiate and manage I/O operations to any channel or device available to the system or LPAR. Channels, I/O devices, interrupts, and memory are owned by the system (or by the LPAR) and not by any specific processor.

This multiprocessor integration appears simple on the surface, but its implementation is complex. It is also important for maximum performance; the ability of any processor to accept any interrupt sent to the system (or to the LPAR) is especially important.

Each processor in a system (or in an LPAR) has a small private area of memory (8 KB starting at real address 0 and always mapped to virtual address 0) that is unique to that processor. This is the Prefix Storage Area (PSA) and is used for interrupt handling and for error handling. A processor can access another processor's PSA through special programming, although this is normally done only for error recovery purposes. A processor can interrupt other processors by using a special instruction (SIGP, for Signal Processor). Again, this is typically used only for error recovery.

3.6 Disk devices

IBM 3390 disk drives are commonly used on current mainframes. Conceptually, this is a simple arrangement, as shown in Figure 3-7.

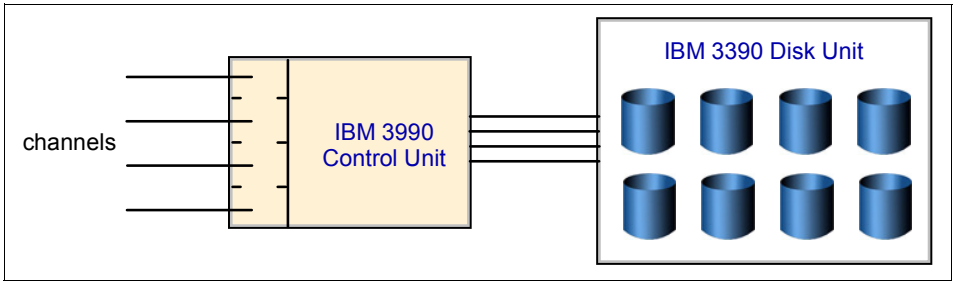


Figure 3-7 Initial IBM 3390 disk implementation

The associated control unit (3990) typically has four channels connected to one or more processors (probably with a switch), and the 3390 unit typically has eight or more disk drives. Each disk drive has the characteristics explained earlier. This illustration shows 3990 and 3390 units, and it also represents the concept or architecture of current devices.

The current equivalent device is an IBM 2105 Enterprise Storage Server®, simplistically illustrated in Figure 3-8.

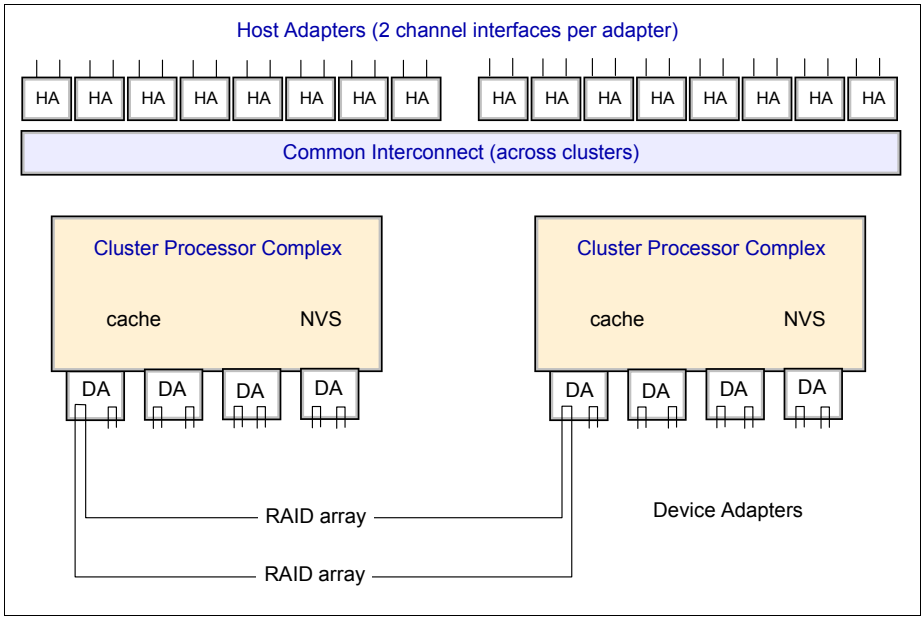


Figure 3-8 Current 3390 implementation

The 2105 unit is a very sophisticated device. It emulates a large number of control units and 3390 disk drives. It contains up to 11 TB of disk space, has up to 32 channel interfaces, 16 GB cache, and 284 MB of non-volatile memory (used for write queueing). The Host Adapters appear as control unit interfaces and can connect up to 32 channels (ESCON or FICON).

The physical disk drives are commodity SCSI-type units (although a serial interface, known as SSA, is used to provide faster and redundant access to the disks). A number of internal arrangements are possible, but the most common involves many RAID 5 arrays with hot spares. Practically everything in the unit has a spare or fallback unit. The internal processing (to emulate 3990 control units and 3390 disks) is provided by four high-end RISC processors in two processor complexes; each complex can operate the total system. Internal batteries preserve transient data during short power failures. A separate console is used to configure and manage the unit.

The 2105 offers many functions not available in real 3390 units, including FlashCopy®, Extended Remote Copy, Concurrent Copy, Parallel Access Volumes, Multiple Allegiance, a huge cache, and so forth.

A simple 3390 disk drive (with control unit) has different technology from the 2105 just described. However, the basic architectural appearance to software is the same. This allows applications and system software written for 3390 disk drives to use the newer technology with no revisions.[11]

There have been several stages of new technology implementing 3390 disk drives; the 2105 is the most recent of these. The process of implementing an architectural standard (in this case the 3390 disk drive and associated control unit) with newer and different technology while maintaining software compatibility is characteristic of mainframe development. As has been mentioned several times, maintaining application compatibility over long periods of technology change is an important characteristic of mainframes.

3.7 Clustering

Clustering has been done on mainframes since the early S/360 days, although the term *cluster* is seldom used there. A clustering technique can be as simple as a shared DASD configuration where manual control or planning is needed to prevent unwanted data overlap.

Additional clustering techniques have been added over the years. In the following paragraphs we discuss three levels of clustering: Basic Shared DASD, CTC rings, and Parallel Sysplex. Most z/OS installations today use one or more of these levels; a single z/OS installation is relatively rare.

In this discussion we use the term "image." A z/OS system (with one or more processors) is a *z/OS image*. A z/OS image might exist on a S/390 or zSeries server (with LPARs), or it might exist in an LPAR, or it might run under z/VM (a hypervisor operating system). A system with six LPARs—each a separate z/OS system—has six z/OS images. We use the term image to indicate that we do not care where (basic system, LPAR, z/VM) a z/OS system is running.

3.7.1 Basic shared DASD

A basic shared DASD environment is illustrated in Figure 3-9. The figure shows z/OS images, but these could be any earlier version of the operating system. This could be two LPARs in the same system or two separate systems; there is absolutely no difference in the concept or operation.

[11] Some software enhancements are needed to use some of the new functions, but these are compatible extensions at the operating system level and do not affect application programs.

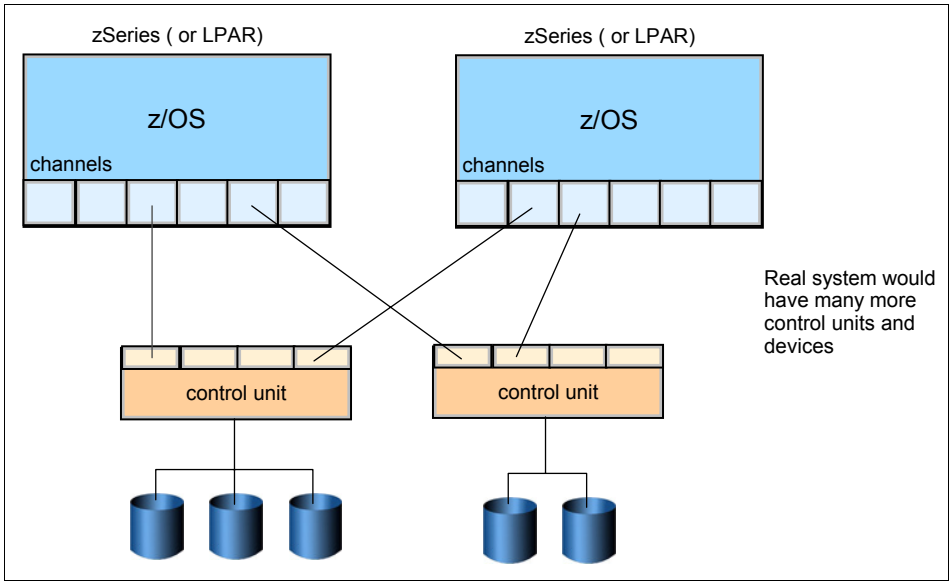


Figure 3-9 Basic shared DASD

The capabilities of a basic shared DASD system are limited. The operating systems automatically issue RESERVE and RELEASE commands to a DASD before updating the volume table of contents (VTOC) or catalog. (The VTOC and catalog are structures that contain metadata for the DASD, indicating where various data sets reside.) The RESERVE command limits access to the entire DASD to the system issuing the command, and this lasts until a RELEASE command is issued. These commands work well for limited periods (such as updating metadata). Applications can also issue RESERVE/RELEASE commands to protect their data sets for the duration of the application. This is not automatically done in this environment and is seldom done in practice because it would lock out other systems' access to the DASD for too long.

A basic shared DASD system is typically used where the Operations staff controls which jobs go to which system and ensures that there is no conflict such as both systems trying to update the same data at the same time. Despite this limitation, a basic shared DASD environment is very useful for testing, recovery, and careful load balancing.

Other types of devices or control units can be attached to both systems. For example, a tape control unit, with multiple tape drives, can be attached to both systems. In this configuration the operators can then allocate individual tape drives to the systems as needed.

3.7.2 CTC rings

Figure 3-10 shows the next level of clustering. This has the same shared DASD as discussed previously, but also has two *channel-to-channel* (CTC) connections between the systems. This is known as a *CTC ring*. (The ring aspect is more obvious when more than two systems are involved.)

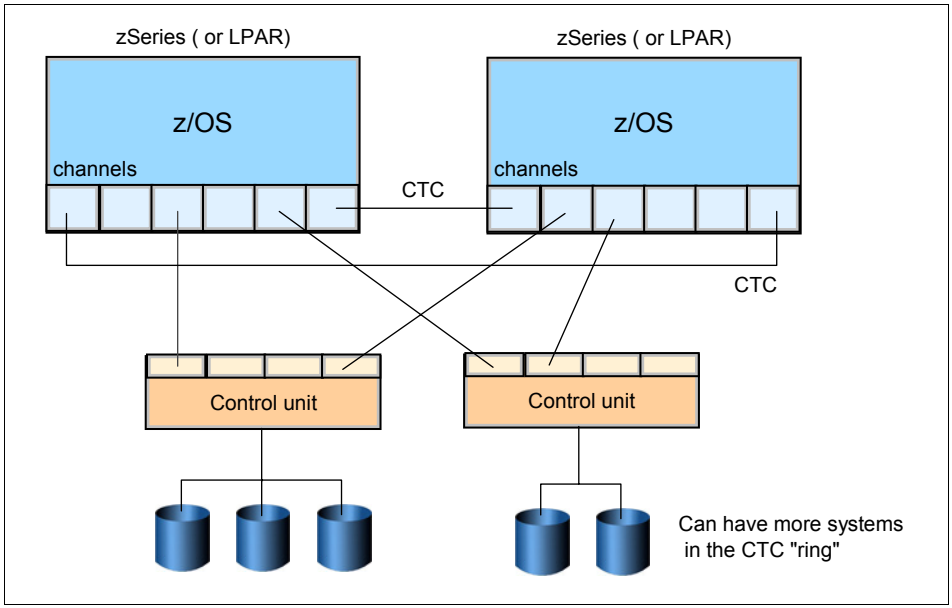


Figure 3-10 Basic sysplex

z/OS can use the CTC ring to pass control information among all systems in the ring. The information that can be passed this way includes:

- Usage and locking information for data sets on disks. This allows the system to automatically prevent unwanted duplicate access to data sets. This locking is based on JCL specifications provided for jobs sent to the system.
- Job queue information such that all the systems in the ring can accept jobs from a single input queue. Likewise, all systems can send printed output to a single output queue.
- Security controls that allow uniform security decisions across all systems.
- Disk metadata controls so that RESERVE and RELEASE disk commands are not necessary.

To a large extent, batch jobs and interactive users can run on any system in this configuration because all disk data sets can be accessed from any z/OS image. Jobs (and interactive users) can be assigned to whichever system is most lightly loaded at the time.

When the CTC configurations were first used, the basic control information shared was locking information. The z/OS component doing this is called global resource serialization function; this configuration is called a GRS ring. The primary limitation of a GRS ring is the latency involved in sending messages around the ring.

A different CTC configuration was used before the ring technique was developed. This required two CTC connections from every system to every other system in the configuration. When more than two or three systems were involved, this became complex and required a considerable number of channels.

The earlier CTC configurations (every-system-to-every-system or a ring configuration) were later developed into a basic *sysplex* configuration. This includes control data sets on the shared DASD. These are used for consistent operational specifications for all systems and to retain information over system restarts.

Configurations with shared DASD, CTC connections, and shared job queues are known as *loosely coupled systems*. (Multiprocessors, where several processors are used by the operating system, are sometimes contrasted as *tightly coupled systems* but this terminology is seldom used. These are also known as Symmetrical MultiProcessors (SMPs); the SMP terminology is common with RISC systems, but is not normally used for mainframes.)

3.7.3 Parallel sysplex

The most recent cluster configuration is a Parallel Sysplex. This involves one or more Coupling Facilities (CFs). A Coupling Facility is a mainframe processor, with memory and special channels, and a built-in operating system. It has no I/O devices, other than the special channels, and the operating system is very small.[12]

A CF functions largely as a fast scratch pad. It is used for three purposes:

- Locking information that is shared among all attached systems
- Cache information (such as for a data base) that is shared among all attached systems
- Data list information that are shared among all attached systems

The information in the CF resides in memory and a CF typically has a large memory. A CF can be a separate system or an LPAR can be used as a CF. Figure 3-11 illustrates a small Parallel Sysplex with two z/OS images. Again, this whole configuration could be in three LPARs of a single system, in three separate systems, or in a mixed combination.

[12] The CF operating system is nothing like z/OS and has no direct user interfaces.

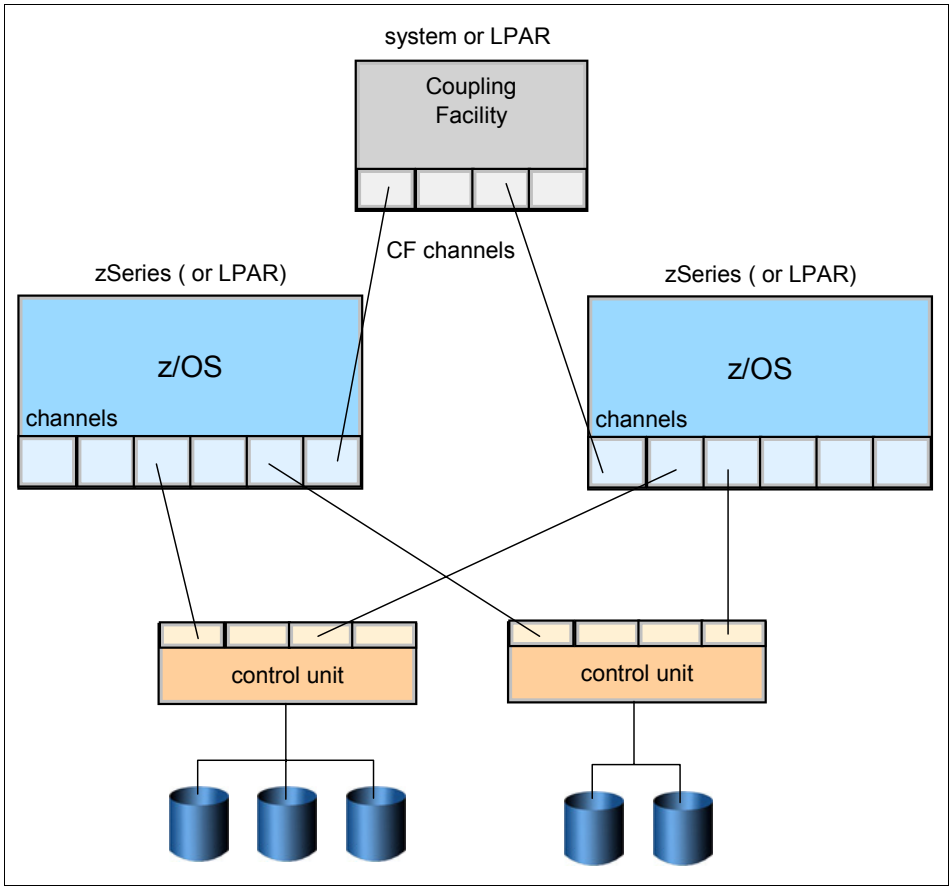


Figure 3-11 Parallel Sysplex

In many ways a Parallel Sysplex system appears as a single large system. It has a single operator interface (which controls all systems). With proper planning and operation (neither of which is trivial), complex workloads can be shared by any or all systems in the Parallel Sysplex, and recovery (by another system in the Parallel Sysplex) can be automatic for many workloads.

Parallel Sysplex systems are described in more detail in Chapter 4, “Parallel Sysplex and continuous availability” on page 103. The purpose of this brief introduction is to introduce additional terminology.

3.8 Typical mainframe systems

We outline the general configurations of three different levels of configuration in this section. These are not intended to be detailed descriptions, but are simply overviews.

3.8.1 Very small systems

The first two examples, in Figure 3-12 on page 97, show that *mainframe* refers more to a *style* of computing rather than to unique hardware. Two different systems are illustrated and neither uses mainframe hardware in the generally accepted sense of the term.

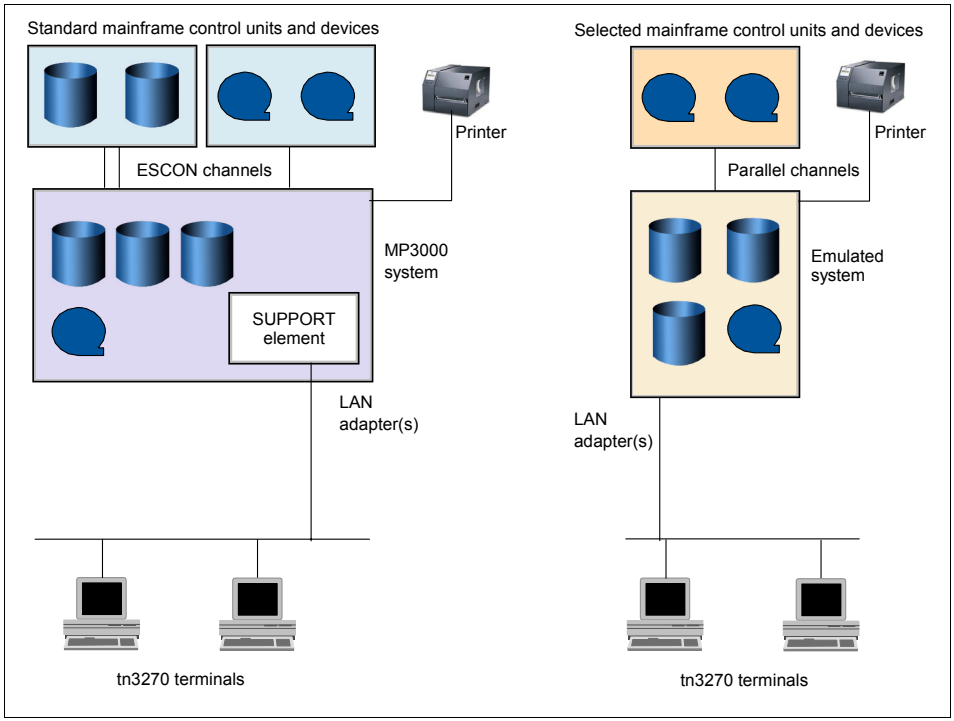


Figure 3-12 Very small mainframe configurations

The first system illustrated is an IBM Multiprise® 3000 system (MP3000). which IBM withdrew from marketing as this book was written. It was the smallest S/390 system produced in recent years. The MP3000 has one or two S/390 processors plus a SAP processor. It also has internal disk drives that can be configured to operate as normal IBM 3390 disk drives. A minimal internal tape drive is normally used for software installation. The MP3000 can have a substantial number of ESCON or parallel channels for connection to traditional external I/O devices.

The MP3000 is completely compatible with S/390 mainframes, but lacks later zSeries features. It can run early versions of z/OS and all prior versions of the operating system. It is typically used with z/VM or z/VSE operating systems.

The second system shown, the emulated zSeries system, has no mainframe hardware. It is based on a personal computer (running Linux or UNIX) and uses software to emulate z/OS[13]. Special PCI channel adapters can be used to connect to selected mainframe I/O devices. The personal computer running the emulated z/OS can have substantial internal disks (typically in a RAID array) for emulating IBM 3390 disk drives.

Both of these systems lack some features found in "real" mainframes. Nevertheless, both are capable of doing quality work. Typical application software cannot distinguish these systems from real mainframes. In fact, these are considered mainframes because their operating systems, their middleware, their applications, and their style of usage are the same as for larger mainframes. The MP3000 can be configured with LPARs and might run both test and production systems. The emulated system does not provide LPARs, but can accomplish much the same thing by running multiple copies of the emulator software.

A key attraction of these systems is that they can be a "mainframe in a box." In many cases no external traditional I/O devices are needed. This greatly reduces the entry-level price for a mainframe system.

3.8.2 Medium single systems

Figure 3-13 on page 99 shows a modest mainframe system and shows the typical external elements needed. The particular system shown is an IBM z890 system with two recent external disk controllers, a number of tape drives, printers, LAN attachments, and consoles.

This is a somewhat idealized configuration in that no older devices are involved. The systems outlined here might have a number of LPARs active, for example:

- A production z/OS system running interactive applications.
- A second production z/OS devoted to major batch applications. (These could also be run in the first LPAR, but some installations prefer a separate LPAR for management purposes.)
- A test z/OS version for testing new software releases, new applications, and so forth.
- One or several Linux partitions, perhaps running Web-related applications.

13

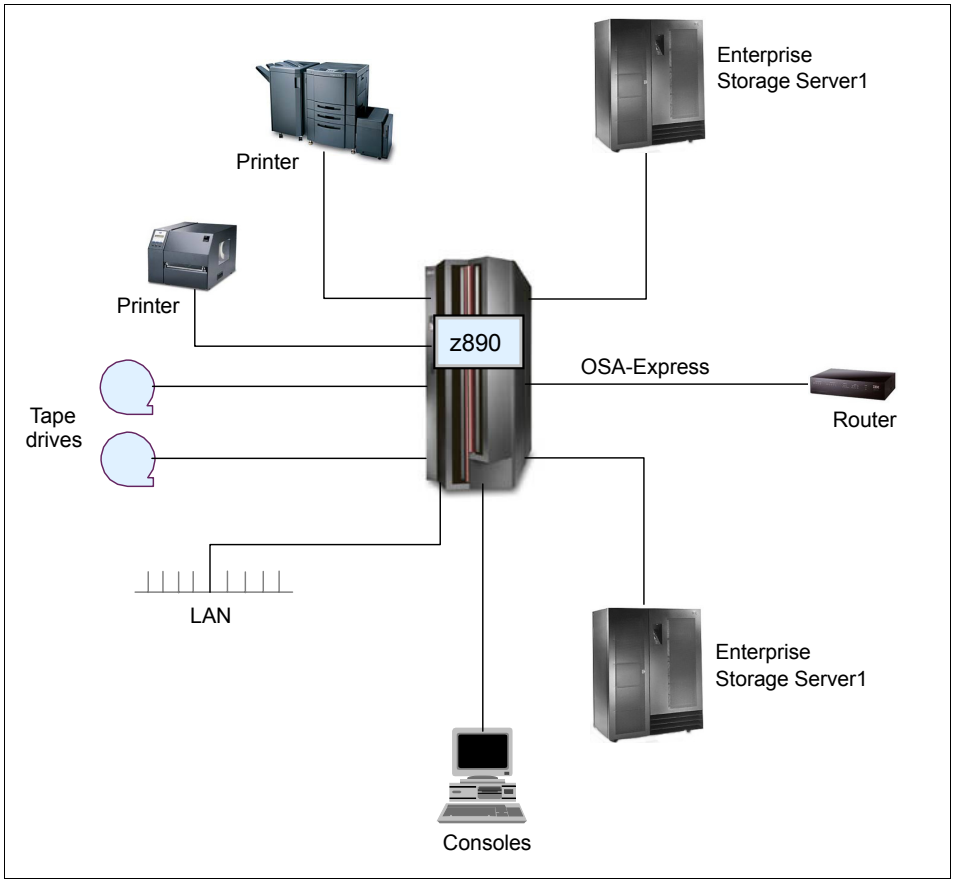


Figure 3-13 Medium mainframe configuration

The disk controllers in Figure 3-13 contain a large number of commodity drives running in multiple RAID configurations. The control unit transforms their interfaces to appear as standard IBM 3390 disk drives, which is the most common disk appearance for mainframes. These disk control units have multiple channel interfaces and can all operate in parallel.

3.8.3 Larger systems

Figure 3-14 shows a larger mainframe, although this is still a modest configuration when compared to a *large* mainframe installation. This example is typical in that both older and newer mainframes are present, along with channel switches allowing all systems to access most I/O devices. Likewise, new and older disk controllers (and devices) and tape controllers (and devices) are present. The total system is in a modest Parallel Sysplex configuration.

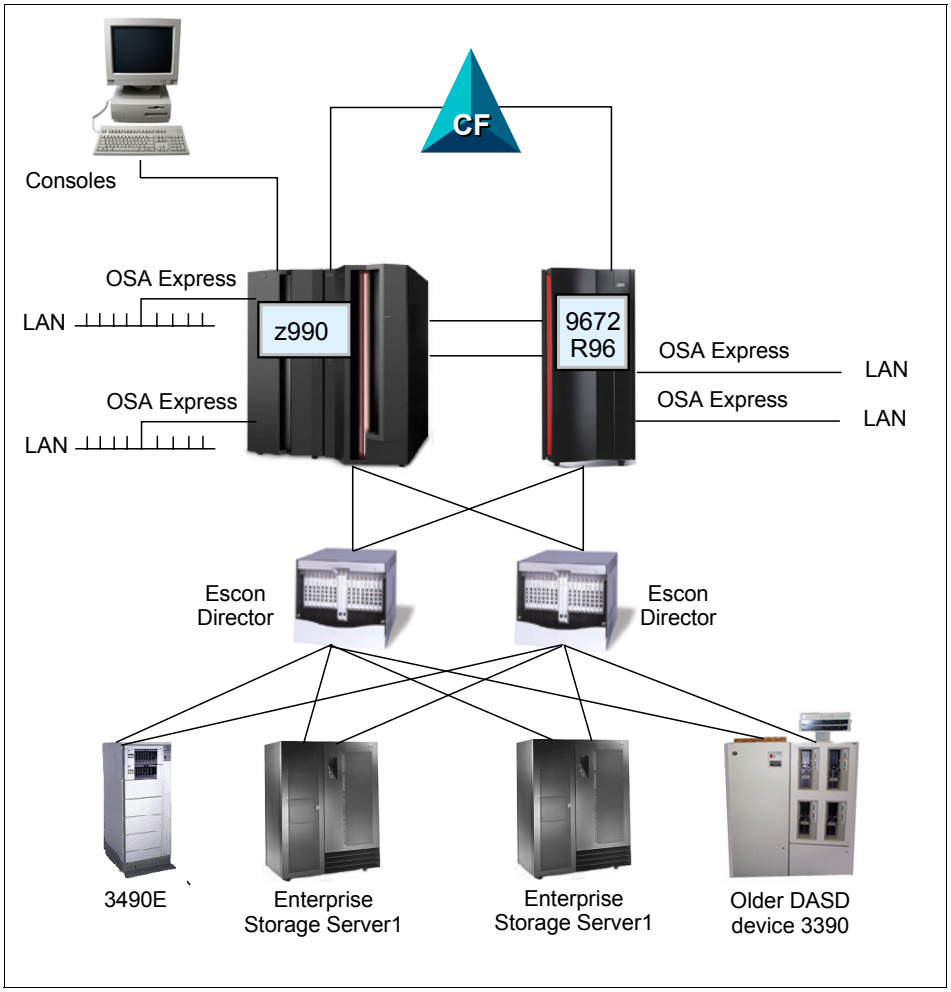


Figure 3-14 Moderately large mainframe configuration

Briefly, the devices in Figure 3-14 include:

- An IBM 3745, which is a communications controller optimized for connection to remote terminals and controllers, and LANs. A 3745 appears as a control unit to the mainframe.
- IBM 3490E tape drives, which, though somewhat outdated, handle the most widely used mainframe-compatible tape cartridges.
- A sixth-generation mainframe design (G6).
- A newer z990 mainframe.
- An Enterprise Storage Server (ESS).
- ESCON directors.

- OSA Express connections to several LANs.
- CF (shown as a separate box, but it might be an LPAR in the mainframe).

3.9 Summary

Being aware of various meanings of the terms *systems*, *processors*, *CP*s, and so forth is important for your understanding of mainframe computers. The original S/360 architecture, based on CPUs, memory, channels, control units, and devices, and the way these are addressed, is fundamental to understanding mainframe hardware—even though almost every detail of the original design has been changed in various ways. The concepts and terminology of the original design still permeate mainframe descriptions and designs.

The ability to partition a large system into multiple smaller systems (LPARs) is now a core requirement in practically all mainframe installations. The flexibility of the hardware design, allowing any processor (CP) to access and accept interrupts for any channel, control unit, and device connected to a given LPAR, contributes to the flexibility, reliability, and performance of the complete system. The availability of a pool of processors (PUs) that can be configured (by IBM) as customer processors (CPs), I/O processors (SAPs), dedicated Linux processors (IFLs), dedicated Java-type processors (zAAPs), and spare processors is unique to mainframes and, again, provides great flexibility in meeting customer requirements. Some of these requirements are based on the cost structures of some mainframe software.

In addition to the primary processors just mentioned (the PUs, and all their characterizations), mainframes have a network of controllers (special microprocessors) that control the system as a whole. These controllers are not visible to the operating system or application programs.

Since the early 1970s mainframes have been designed as multiprocessor systems, even when only a single processor is installed. All operating system software is designed for multiple processors; a system with a single processor is considered a special case of a general multiprocessor design.

All but the smallest mainframe installations typically use clustering techniques, although they do not normally use the terms *cluster* or *clustering*. As stated previously, a clustering technique can be as simple as a shared DASD configuration where manual control or planning is needed to prevent unwanted data overlap.

More common today are configurations that allow sharing of locking and enqueueing controls among all systems. Among other benefits, this automatically manages access to data sets so that unwanted concurrent usage does not occur.

The most sophisticated of the clustering techniques is a Parallel Sysplex, which is discussed in Chapter 4, “Parallel Sysplex and continuous availability” on page 103.

Key terms in this topic		
central electronic complex (CEC)	central processing unit (CPU)	Channel Path Identifier (CHPID)
channel-to-channel (CTC)	ESCON channel	FICON channel
hardware management console (HMC)	I/O connectivity	Integrated Facility for Linux (IFL)
logical partition (LPAR)	multiprocessor	power-on reset (POR)
Support Element (SE)	unicode	zAAP processor

4

Parallel Sysplex and continuous availability

Objective: In working with the z/OS operating system, you need to understand how it achieves near-continuous availability through technologies such as "no single points of failure."

After completing this topic, you will be able to:

- Discuss Parallel Sysplex.
- Explain how Parallel Sysplex can achieve continuous availability.
- Explain dynamic workload balancing.
- Explain the single system image.

4.1 Future of the new mainframe

This course concludes with a discussion of Parallel Sysplex and some examples of how this powerful technology is used to ensure continuous availability in today's large systems environments.

Parallel Sysplex technology allows the linking up to 32 servers with near linear scalability to create a powerful commercial processing clustered system. Every server in a Parallel Sysplex cluster can be configured to share access to data resources, and a "cloned" instance of an application might run on every server.

Parallel Sysplex design characteristics help businesses to run continuously, even during periods of dramatic change. Sysplex sites can dynamically add and change systems in a sysplex, and configure the systems for no single points of failure.

Through this state-of-the-art cluster technology, multiple z/OS systems can be made to work in concert to more efficiently process the largest commercial workloads.

4.2 What a Parallel Sysplex is

A *sysplex* is a collection of z/OS systems that cooperate, using certain hardware and software products, to process work. It is a clustering technology that can provide near-continuous availability.

A conventional large computer system also uses hardware and software products that cooperate to process work. A major difference between a sysplex and a conventional large computer system is the improved growth potential and level of availability in a sysplex. The sysplex increases the number of processing units and z/OS operating systems that can cooperate, which in turn increases the amount of work that can be processed. To facilitate this cooperation, new products were developed and old products were enhanced.

A *Parallel Sysplex* is a sysplex that uses multisystem data-sharing technology. It allows direct, concurrent read/write access to shared data from all processing nodes (or servers) in the configuration without impacting performance or data integrity. Each node can concurrently cache shared data in local processor memory through hardware-assisted cluster-wide serialization and coherency controls.

As a result, work requests that are associated with a single workload, such as business transactions or database queries, can be dynamically distributed for parallel execution on nodes in the sysplex cluster based on available processor capacity.

Figure 4-1 shows the visible parts of a Parallel Sysplex, namely the hardware. These are the key components of Parallel Sysplex as implemented in the hardware architecture.

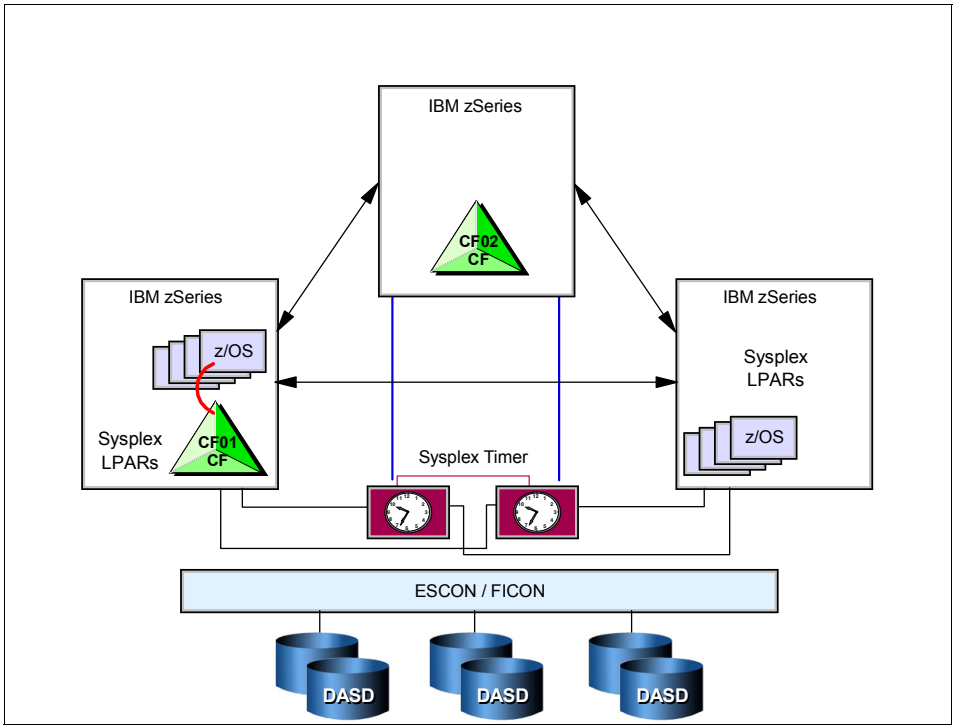


Figure 4-1 Sysplex hardware overview

4.2.1 Shared data clustering

Parallel Sysplex technology extends the strengths of IBM mainframe computers by linking up to 32 servers with near linear scalability to create a powerful commercial processing clustered system. Every server in a Parallel Sysplex cluster has access to all data resources, and every “cloned” application can run on every server. Using mainframe Coupling Technology, Parallel Sysplex technology provides a “shared data” clustering technique that permits multi-system data sharing with high performance read/write integrity.

This “shared data” (as opposed to “shared nothing”) approach enables workloads to be dynamically balanced across servers in the Parallel Sysplex cluster. It enables critical business applications to take advantage of the aggregate capacity of multiple servers to help ensure maximum system throughput and performance during peak processing periods. In the event of a hardware or software outage, either planned or unplanned, workloads can be dynamically redirected to available servers, thus providing near-continuous application availability.

4.2.2 Non-disruptive maintenance

Another unique advantage of using Parallel Sysplex technology is the ability to perform hardware and software maintenance and installation in a non-disruptive manner.

Through data sharing and dynamic workload management, servers can be dynamically removed from or added to the cluster, allowing installation and maintenance activities to be performed while the remaining systems continue to process work. Furthermore, by adhering to the IBM software and hardware coexistence policy, software and/or hardware upgrades can be introduced one system at a time. This capability allows customers to roll changes through systems at a pace that makes sense for their business.

The ability to perform rolling hardware and software maintenance in a non-disruptive manner allows business to implement critical business function and react to rapid growth without affecting customer availability.

4.3 Continuous availability of mainframes

Parallel Sysplex technology is an enabling technology, allowing highly reliable, redundant, and robust mainframe technologies to achieve near-continuous availability. A properly configured Parallel Sysplex cluster is designed to remain available to its users and applications with minimal downtime, for example:

- Hardware and software components provide for concurrency to facilitate non-disruptive maintenance, like Capacity Upgrade on Demand, which allows processing or coupling capacity to be added one engine at a time without disruption to running workloads.
- DASD subsystems employ disk mirroring or RAID technologies to help protect against data loss, and exploit technologies to enable point-in-time backup, without the need to shut down applications.
- Networking technologies deliver functions such as VTAM Generic Resources, Multi-Node Persistent Sessions, Virtual IP Addressing, and Sysplex Distributor to provide fault-tolerant network connections.
- I/O subsystems support multiple I/O paths and dynamic switching to prevent loss of data access and improved throughput.
- z/OS software components allow new software releases to coexist with lower levels of those software components to facilitate rolling maintenance.
- Business applications are “data sharing-enabled” and cloned across servers to allow workload balancing to prevent loss of application availability in the event of an outage.
- Operational and recovery processes are fully automated and transparent to users, and reduce or eliminate the need for human intervention.

Parallel Sysplex is a way of managing this multi-system environment, providing such benefits as:

- "No single points of failure" on page 107
- "Capacity and scaling" on page 108
- "Dynamic workload balancing" on page 108
- "Ease of use" on page 109
- "Single system image" on page 111
- "Compatible change and non-disruptive growth" on page 112
- "Application compatibility" on page 112
- "Disaster recovery" on page 113

These benefits are described in the remaining sections of this topic.

4.3.1 No single points of failure

In a Parallel Sysplex cluster, it is possible to construct a parallel processing environment with no single points of failure. Because all of the systems in the Parallel Sysplex can have concurrent access to all critical applications and data, the loss of a system due to either hardware or software failure does not necessitate loss of application availability.

Peer instances of a failing subsystem executing on remaining healthy system nodes can take over recovery responsibility for resources held by the failing instance. Alternatively, the failing subsystem can be automatically restarted on still-healthy systems using automatic restart capabilities to perform recovery for work in progress at the time of the failure. While the failing subsystem instance is unavailable, new work requests can be redirected to other data-sharing instances of the subsystem on other cluster nodes to provide continuous application availability across the failure and subsequent recovery. This provides the ability to mask planned as well as unplanned outages to the end user.

Because of the redundancy in the configuration, there is a significant reduction in the number of single points of failure. Without a Parallel Sysplex, the loss of a server could severely impact the performance of an application, as well as introduce system management difficulties in redistributing the workload or reallocating resources until the failure is repaired. In an Parallel Sysplex environment, it is possible that the loss of a server may be transparent to the application, and the server workload can be redistributed automatically within the Parallel Sysplex with little performance degradation. Therefore, events that otherwise would seriously impact application availability, such as failures in CEC hardware elements or critical operating system components, would, in a Parallel Sysplex environment, have reduced impact.

Even though they work together and present a single image, the nodes in a Parallel Sysplex cluster remain individual systems, making installation, operation, and maintenance non-disruptive. The system programmer can introduce changes, such as software upgrades, one system at a time, while the remaining systems continue to process

work. This allows the mainframe IT staff to roll changes through its systems on a schedule that is convenient to the business.

4.3.2 Capacity and scaling

The Parallel Sysplex environment can scale nearly linearly from 2 to 32 systems. This can be a mix of any servers that support the Parallel Sysplex environment. The aggregate capacity of this configuration meets every processing requirement known today.

4.3.3 Dynamic workload balancing

The entire Parallel Sysplex cluster can be viewed as a single logical resource to end users and business applications. Just as work can be dynamically distributed across the individual processors within a single SMP server, so too can work be directed to any node in a Parallel Sysplex cluster having available capacity. This avoids the need to partition data or applications among individual nodes in the cluster or to replicate databases across multiple servers.

Workload balancing also permits a business to run diverse applications across a Parallel Sysplex cluster while maintaining the response levels critical to a business. The mainframe IT director selects the service level agreements required for each workload, and the workload management (WLM) component of z/OS, along with subsystems such as CP/SM or IMS, automatically balances tasks across all the resources of the Parallel Sysplex cluster to meet these business goals. The work can come from a variety of sources, such as batch, SNA, TCP/IP, DRDA, or WebSphere MQ.

There are several aspects to consider for recovery. First, when a failure occurs, it is important to bypass it by automatically redistributing the workload to utilize the remaining available resources. Secondly, it is necessary to recover the elements of work that were in progress at the time of the failure. Finally, when the failed element is repaired, it should be brought back into the configuration as quickly and transparently as possible to again start processing the workload. Parallel Sysplex technology enables all this to happen.

Workload distribution

After the failing element has been isolated, it is necessary to non-disruptively redirect the workload to the remaining available resources in the Parallel Sysplex. In the event of failure in the Parallel Sysplex environment, the online transaction workload is automatically redistributed without operator intervention.

Generic resource management

Generic resource management provides the ability to specify to VTAM a common network interface. This can be used for CICS terminal owning regions (TORs), IMS Transaction Manager, TSO, or DB2 DDF work. If one of the CICS TORs fails, for

example, only a subset of the network is affected. The affected terminals are able to immediately log on again and continue processing after being connected to a different TOR.

4.3.4 Ease of use

The Parallel Sysplex solution satisfies a major customer requirement for continuous 24-hour-a-day, 7-day-a-week availability, while providing techniques for achieving simplified Systems Management consistent with this requirement. Some of the features of the Parallel Sysplex solution that contribute to increased availability also help to eliminate some Systems Management tasks. Examples include:

- “Workload management (WLM) component” on page 109
- “Sysplex Failure Manager (SFM)” on page 109
- “Automatic Restart Manager (ARM)” on page 110
- “Cloning and symbolics” on page 110
- “zSeries resource sharing” on page 110

Workload management (WLM) component

The workload management (WLM) component of z/OS provides sysplex-wide workload management capabilities based on installation-specified performance goals and the business importance of the workloads. WLM tries to attain the performance goals through dynamic resource distribution. WLM provides the Parallel Sysplex cluster with the intelligence to determine where work needs to be processed and in what priority. The priority is based on the customer's business goals and is managed by sysplex technology.

Sysplex Failure Manager (SFM)

The Sysplex Failure Management policy allows the installation to specify failure detection intervals and recovery actions to be initiated in the event of the failure of a system in the sysplex.

Without SFM, when one of the systems in the Parallel Sysplex fails, the operator is notified and prompted to take some recovery action. The operator may choose to partition the non-responding system from the Parallel Sysplex, or to take some action to try to recover the system. This period of operator intervention might tie up critical system resources required by the remaining active systems. Sysplex Failure Manager allows the installation to code a policy to define the recovery actions to be initiated when specific types of problems are detected, such as fencing off the failed image that prevents access to shared resources, logical partition deactivation, or central storage and expanded storage acquisition, to be automatically initiated following detection of a Parallel Sysplex failure.

Automatic Restart Manager (ARM)

Automatic restart manager enables fast recovery of subsystems that might hold critical resources at the time of failure. If other instances of the subsystem in the Parallel Sysplex need any of these critical resources, fast recovery will make these resources available more quickly. Even though automation packages are used today to restart the subsystem to resolve such deadlocks, ARM can be activated closer to the time of failure.

Automatic Restart Manager reduces operator intervention in the following areas:

- Detection of the failure of a critical job or started task
- Automatic restart after a started task or job failure

  After an ABEND of a job or started task, the job or started task can be restarted with specific conditions, such as overriding the original JCL or specifying job dependencies, without relying on the operator.
- Automatic redistribution of work to an appropriate system following a system failure

  This removes the time-consuming step of human evaluation of the most appropriate target system for restarting work

Cloning and symbolics

Cloning refers to replicating the hardware and software configurations across the different physical servers in the Parallel Sysplex. That is, an application that is going to take advantage of parallel processing might have identical instances running on all images in the Parallel Sysplex. The hardware and software supporting these applications could also be configured identically on all systems in the Parallel Sysplex to reduce the amount of work required to define and support the environment.

The concept of *symmetry* allows new systems to be introduced and enables automatic workload distribution in the event of failure or when an individual system is scheduled for maintenance. It also reduces the amount of work required by the system programmer in setting up the environment. Note that symmetry does *not* preclude the need for systems to have unique configuration requirements, such as the asymmetric attachment of printers and communications controllers, or asymmetric workloads that do not lend themselves to the parallel environment.

System symbolics are used to help manage cloning. z/OS provides support for the substitution values in startup parameters, JCL, system commands, and started tasks. These values can be used in parameter and procedure specifications to allow unique substitution when dynamically forming a resource name.

zSeries resource sharing

A number of base z/OS components have discovered that IBM S/390 Coupling Facility shared storage provides a medium for sharing component information for the purpose of multi-system resource management. This exploitation, called IBM zSeries Resource

Sharing, enables sharing of physical resources such as files, tape drives, consoles, and catalogs with improvements in cost, performance and simplified systems management. This is *not to be confused* with Parallel Sysplex data sharing by the database subsystems. zSeries Resource Sharing delivers immediate value even for customers who are not leveraging data sharing, through native system exploitation delivered with the base z/OS software stack.

One of the goals of the Parallel Sysplex solution is to provide simplified systems management by reducing complexity in managing, operating, and servicing a Parallel Sysplex, without requiring an increase in the number of support staff and without reducing availability.

4.3.5 Single system image

Even though there could be multiple servers and z/OS images in the Parallel Sysplex and a mix of different technologies, the collection of systems in the Parallel Sysplex should appear as a single entity to the operator, the end user, the database administrator, and so on. A single system image brings reduced complexity from both operational and definition perspectives.

Regardless of the number of system images and the complexity of the underlying hardware, the Parallel Sysplex solution provides for a single system image from several perspectives:

- Data access, allowing dynamic workload balancing and improved availability
- Dynamic Transaction Routing, providing dynamic workload balancing and improved availability
- End-user interface, allowing logon to a logical network entity
- Operational interfaces, allowing easier Systems Management

Single point of control

It is a requirement that the collection of systems in the Parallel Sysplex can be managed from a logical single point of control. The term “single point of control” means the ability to access whatever interfaces are required for the task in question, without reliance on a physical piece of hardware. For example, in a Parallel Sysplex of many systems, it is necessary to be able to direct commands or operations to any system in the Parallel Sysplex, without the necessity for a console or control point to be physically attached to every system in the Parallel Sysplex.

Persistent single system image across failures

Even though individual hardware elements or entire systems in the Parallel Sysplex fail, a single system image must be maintained. This means that, as with the concept of single point of control, the presentation of the single system image is not dependent on a specific physical element in the configuration. From the end-user point of view, the

parallel nature of applications in the Parallel Sysplex environment must be transparent. An application should be accessible regardless of which physical z/OS image supports it.

4.3.6 Compatible change and non-disruptive growth

A primary goal of Parallel Sysplex is continuous availability. Therefore, it is a requirement that changes such as new applications, software, or hardware can be introduced non-disruptively, and that they be able to coexist with current levels. In support of compatible change, the hardware and software components of the Parallel Sysplex solution will allow the coexistence of two levels, that is, level N and level N+1. This means, for example, that no IBM software product will make a change that cannot be tolerated by the previous release.

4.3.7 Application compatibility

A design goal of Parallel Sysplex clustering is that no application changes be required to take advantage of the technology. For the most part, this has held true, although some affinities need to be investigated to get the maximum advantage from the configuration.

From the application architects' point of view, three major points might lead to the decision to run an application in a Parallel Sysplex:

- Technology benefits

  Scalability (even with non-disruptive upgrades), availability, and dynamic workload management are tools that enable an architect to meet customer needs in cases where the application plays a key role in the customer's business process. With the multisystem data sharing technology, all processing nodes in a Parallel Sysplex have full concurrent read/write access to shared data without affecting integrity and performance.

- Integration benefits

  Since many applications are historically S/390- and z/OS-based, new applications on z/OS get performance and maintenance benefits, especially if they are connected to existing applications.

- Infrastructure benefits

  If there is already an existing Parallel Sysplex, it needs very little infrastructure work to integrate a new application. In many cases the installation does not need to integrate new servers. Instead it can leverage the existing infrastructure and make use of the strengths of the existing sysplex. With Geographically Dispersed Parallel Sysplex™ (GDPS®)—connecting multiple sysplexes in different locations—the mainframe IT staff can create a configuration that is enabled for disaster recovery.

4.3.8 Disaster recovery

Geographically Dispersed Parallel Sysplex (GDPS) is the primary disaster recovery and continuous availability solution for a mainframe-based multi-site enterprise. GDPS automatically mirrors critical data and efficiently balances workload between the sites.

GDPS also uses automation and Parallel Sysplex technology to help manage multi-site databases, processors, network resources and storage subsystem mirroring. This technology offers continuous availability, efficient workload management, resource management and prompt data recovery for business-critical mainframe applications and data. With GDPS, the current maximum distance between the two sites is 100km (about 62 miles) of fiber, although there are some other restrictions. This provides a synchronous solution that helps to ensure no loss of data.

There is also GDPS/XRC, which can be used over extended distances and should provide a recovery point objective of less than two minutes (that is, a maximum of two minutes of data would need to be recovered or is lost).

4.4 Summary

Parallel Sysplex technology allows the linking up to 32 servers with near linear scalability to create a powerful commercial processing clustered system. Every server in a Parallel Sysplex cluster has access to all data resources, and every “cloned” application can run on every server. When used with coupling technology, Parallel Sysplex provides a “shared data” clustering technique that permits multi-system data sharing with high performance read/write integrity.

Sysplex design characteristics help businesses to run continuously, even during periods of dramatic change. Sysplex sites can dynamically add and change systems in a sysplex, and configure the systems for no single points of failure.

Through this state-of-the-art cluster technology, multiple z/OS systems can be made to work in concert to more efficiently process the largest commercial workloads.

Key terms in this topic		
Automatic Restart Manager (ARM)	cloning	continuous availability
coupling technology	dynamic workload management	Geographically Dispersed Parallel Sysplex (GDPS)
Parallel Sysplex	shared data clustering	single point of control
symbolics	sysplex	zSeries resource sharing