

z/OS



MVS Initialization and Tuning Guide

z/OS



MVS Initialization and Tuning Guide

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page B-1.

Seventh Edition, April 2008

This is a major revision to SA22-7591-05.

This edition applies to Version 1 Release 9 of z/OS (5694-A01), and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405

FAX (Other Countries):

Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)

Internet e-mail: mhvrdfs@us.ibm.com

World Wide Web: <http://www.ibm.com/servers/eserver/zseries/zos/webqs.html>

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

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

© Copyright International Business Machines Corporation 1991, 2008. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
Tables	vii
About this document	ix
Who should use this document	ix
Where to find more information	ix
Information updates on the web	ix
Summary of changes	xi
Chapter 1. Storage Management Overview	1-1
Initialization Process	1-1
System Address Space Creation	1-2
Master Scheduler Initialization	1-5
Subsystem Initialization	1-5
START/LOGON/MOUNT Processing.	1-5
Processor Storage Overview	1-6
System Preferred Area.	1-7
Nucleus Area	1-8
The Fixed Link Pack Area (FLPA).	1-8
System Queue Area (SQA-Fixed).	1-9
Fixed LSQA Storage Requirements	1-9
V=R Area	1-9
Expanded Storage Overview	1-10
Virtual Storage Overview	1-10
The Virtual Storage Address Space	1-10
General Virtual Storage Allocation Considerations	1-12
System Queue Area (SQA/Extended SQA)	1-12
Pageable Link Pack Area (PLPA/Extended PLPA)	1-14
Placing Modules in the System's Search Order for Programs	1-14
Modified Link Pack Area (MLPA/Extended MLPA)	1-23
Common Service Area (CSA/Extended CSA)	1-24
Local System Queue Area (LSQA/Extended LSQA).	1-25
Scheduler Work Area (SWA/Extended SWA)	1-25
Subpools 229, 230, 249 - Extended 229, 230, 249	1-25
System Region	1-25
The Private Area User Region/Extended Private Area User Region	1-26
Identifying Problems in Virtual Storage (DIAGxx Parmlib Member)	1-29
Auxiliary Storage Overview.	1-30
System Data Sets	1-30
Paging Data Sets	1-32
Improving Module Fetch Performance With LLA	1-33
LLA and Module Search Order	1-34
Planning To Use LLA	1-34
Coding the Required Members of Parmlib	1-35
Controlling LLA and VLF Through Operator Commands	1-37
Allocation Considerations	1-42
Serialization of Resources During Allocation	1-42
Improving Allocation Performance	1-43
The Volume Attribute List	1-43
Use and Mount Attributes	1-44

Chapter 2. Auxiliary Storage Management Initialization	2-1
Page Operations	2-1
Paging Operations and Algorithms	2-1
Page Data Set Sizes	2-2
Page Data Set Protection	2-2
SYSTEMS level ENQ	2-2
Status Information Record	2-3
Space Calculation Examples	2-3
Example 1: Sizing the PLPA Page Data Set, Size of the PLPA and Extended PLPA Unknown	2-3
Example 2: Sizing the PLPA Page Data Set, Size of the PLPA and Extended PLPA Known	2-4
Example 3: Sizing the Common Page Data Set	2-4
Example 4: Sizing Local Page Data Sets	2-4
Performance Recommendations	2-5
Estimating Total Size of Paging Data Sets	2-7
Using Measurement Facilities	2-7
Adding More Paging Space	2-7
Deleting, Replacing or Draining Page Data Sets	2-8
Questions and Answers	2-8
 Chapter 3. The System Resources Manager	 3-1
System Tuning and SRM	3-1
Section 1: Description of the System Resources Manager (SRM)	3-2
Controlling SRM	3-2
Objectives	3-2
Types of Control	3-3
Functions	3-3
I/O Service Units	3-11
Section 2: Basic SRM Parameter Concepts	3-11
MPL Adjustment Control	3-11
Transaction Definition for CLISTs	3-12
Directed VIO Activity	3-12
Alternate Wait Management	3-12
Dispatching Mode Control	3-13
Section 3: Advanced SRM Parameter Concepts	3-13
Selective Enablement for I/O	3-13
Adjustment of Constants Options	3-15
Section 4: Guidelines	3-16
Defining Installation Requirements	3-16
Preparing an Initial OPT	3-17
Section 5: Installation Management Controls	3-25
Operator Commands Related to SRM	3-25
 Appendix. Accessibility	 A-1
Using assistive technologies	A-1
Keyboard navigation of the user interface	A-1
z/OS information	A-1
 Notices	 B-1
Programming Interface Information	B-2
Trademarks	B-2
 Index	 X-1

Figures

1-1.	Virtual Storage Layout for Multiple Address Spaces.	1-4
1-2.	Virtual Storage Layout for a Single Address Space	1-11
1-3.	Auxiliary Storage Requirement Overview	1-31

Tables

1-1.	Offsets for the SQA/CSA threshold levels	1-13
1-2.	FREEZEINOFREEZE Processing	1-40
1-3.	Processing Order for Allocation Requests Requiring Serialization	1-42
1-4.	Summary of Mount and Use Attribute Combinations	1-45
1-5.	Sharable and Nonsharable Volume Requests	1-47
2-1.	Page data set Values.	2-3
3-1.	Relating SRM Seconds to Real Time	3-14
3-2.	IBM System z9 Enterprise Class (z9 EC) Processor Models	3-18
3-3.	zSeries 990 Processor Models	3-19
3-4.	zSeries 900 Processor Models	3-20
3-5.	zSeries 890 Processor Models	3-22
3-6.	zSeries 800 Processor Models	3-22
3-7.	S/390 9672 Processor Models	3-23
3-8.	S/390 3000 processor models	3-24

About this document

This document is a preliminary tuning guide for the MVS™ element of z/OS®. The document describes how to initialize the system and how to get improved system performance.

For information about how to install the software products that are necessary to run z/OS, see *z/OS Planning for Installation*.

Who should use this document

This document is for anyone whose job includes designing and planning to meet installation needs based on system workload, resources, and requirements. For that audience, the document is intended as a guide to what to do to implement installation policies.

The document is also for anyone who tunes the system. This person must be able to determine where the system needs adjustment, to understand the effects of changing the system parameters, and to determine what changes to the system parameters will bring about the desired effect.

Where to find more information

Where necessary, this document references information in other documents, using the shortened version of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS, see the online document at:

<http://www.s390.ibm.com:80/bookmgr-cgi/bookmgr.cmd/BOOKS/ZIDOCMST/CCONTENTS>

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

Summary of changes

Summary of changes for SA22-7591-06 z/OS Version 1 Release 9 as updated April 2008

The document contains information previously presented in *z/OS MVS Initialization and Tuning Guide*, SA22-7591-05, which supports z/OS Version 1 Release 9.

New information:

- The Dispatching Mode Control function. See “Dispatching Mode Control” on page 3-13.
- The NOHONORIEFUSIREGION attribute is used to bypass IEFUSI region controls for specific programs that require a larger than normal region in order to successfully execute. See Chapter 1, “Storage Management Overview,” on page 1-1.

You may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of changes for SA22-7591-05 z/OS Version 1 Release 9

The document contains information previously presented in *z/OS MVS Initialization and Tuning Guide*, SA22-7591-04, which supports z/OS Version 1 Release 8.

Deleted information:

- Expanded storage is no longer supported in z/Architecture® mode, expanded storage information has been removed from this document.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes for SA22-7591-04 z/OS Version 1 Release 8

The document contains information previously presented in *z/OS MVS Initialization and Tuning Guide*, SA22-7591-03, which supports z/OS Version 1 Release 7.

New information:

- Service consumed per second of execution time by System z9™ 109 processor models. See Table 3-2 on page 3-18.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

**Summary of changes
for SA22-7591-03
z/OS Version 1 Release 7**

The document contains information previously presented in *z/OS MVS Initialization and Tuning Guide*, SA22-7591-02, which supports z/OS Version 1 Release 4.

New information:

- New information about how address spaces use system services. See “System Address Space Creation” on page 1-2.
- New information about “The Fixed Link Pack Area (FLPA)” on page 1-8.
- New information about “Fixed LSQA Storage Requirements” on page 1-9.
- New information about “The Virtual Storage Address Space” on page 1-10.
- New section, “SQA/CSA Thresholds” on page 1-13.
- New information about “Pageable Link Pack Area (PLPA/Extended PLPA)” on page 1-14.
- New information about “Modified Link Pack Area (MLPA/Extended MLPA)” on page 1-23.
- New information about recommended common storage tracking in “Identifying Problems in Virtual Storage (DIAGxx Parmlib Member)” on page 1-29.
- New information about “Paging Operations and Algorithms” on page 2-1, “Page Data Set Sizes” on page 2-2, and “Page Data Set Protection” on page 2-2.
- New information about “Space Calculation Examples” on page 2-3.
- New information about “Performance Recommendations” on page 2-5.

Deleted information:

- Beginning with z/OS V1R3, workload manager (WLM) compatibility mode is no longer available. Information about WLM compatibility mode has been removed from this document.

References to OpenEdition have been replaced with z/OS UNIX® System Services or z/OS UNIX.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Chapter 1. Storage Management Overview

To tailor the system's storage parameters, you need a general understanding of the system initialization and storage initialization processes.

This chapter contains the following topics:

- "Initialization Process"
- "Processor Storage Overview" on page 1-6
- "Virtual Storage Overview" on page 1-10
- "Auxiliary Storage Overview" on page 1-30
- "Improving Module Fetch Performance With LLA" on page 1-33
- "Allocation Considerations" on page 1-42.

For information about the storage management subsystem (SMS), see the DFSMS™ library.

Initialization Process

The system initialization process prepares the system control program and its environment to do work for the installation. The 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 operator selects the LOAD function at the system console. MVS locates all of the usable central storage that is online and available to the system, and creates a virtual environment for the building of various system areas.

IPL includes the following major initialization functions:

- Loads the DAT-off nucleus into central storage.
- Loads the DAT-on nucleus into virtual storage so that it spans above and below 16 megabytes (except the prefixed storage area (PSA), which IPL loads at virtual zero).
- Builds the nucleus map, NUCMAP, of the DAT-on nucleus. NUCMAP resides in virtual storage above the nucleus.
- In ESA/390 mode, builds the page frame table (PFT) in virtual storage above the NUCMAP.
- Allocates the system's minimum virtual storage for the system queue area (SQA) and the extended SQA.
- Allocates virtual storage for the extended local system queue area (extended LSQA) for the master scheduler address space.

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. For more information about quick starts and warm starts, see *z/OS MVS Initialization and Tuning Reference*.
- 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.

See Figure 1-1 for the relative position of the system areas in virtual storage. Most of the system areas exist both below and above 16 megabytes, providing an environment that can support both 24-bit and 31-bit addressing. However, each area and its counterpart above 16 megabytes can be thought of as a single logical area in virtual storage.

System Address Space Creation

In addition to initializing system areas, MVS establishes system component address spaces. MVS establishes an address space for the master scheduler (the master scheduler address space) and other system address spaces for various subsystems and system components. Some of the component address spaces are:

MASTER	Master address space
ABARS, ABARxxxx	1 to 15 DFSMSHsm™ secondary address spaces to perform aggregate backup or aggregate recovery processing.
ALLOCAS	Allocation services and data areas
ANTMAIN	Concurrent copy support
APPC	APPC/MVS component
ASCH	APPC/MVS scheduling
CATALOG	Catalog functions. Also known as CAS (catalog address space).
BPXOINIT	z/OS UNIX System Services
CONSOLE	Communications task
DFM	Distributed File Manager/MVS
DFMCAS	Distributed File Manager/MVS
DLF	Data lookaside facility
DUMPSRV	Dumping services
HSM	DFSMSHsm
FTPSEVE	FTP server(s); can be user-specified names.

GDEDFM	For each Distributed File Manager/MVS user conversation that is active, an address space named GDEDFM is created.
GRS	Global resource serialization
IEFSCHAS	Scheduler address space
IOSAS	I/O supervisor, ESCON®, I/O recovery
IXGLOGR	System logger
JES2	JES2
JES2AUX	JES2 additional support
JES2MON	JES2 address space monitor
JES3	JES3
JES3AUX	JES3 additional support
JES3DLOG	JES3 hardcopy log (DLOG)
JESXCF	JES common coupling services address space
LLA	Link list
NFS	DFSMS/MVS™ Network File System address space
OAM	DFSMSdfp™ Object Access Method
OMVS	z/OS UNIX System Services
PCAUTH	Cross-memory support
PORTMAP	Portmapper function
RASP	Real storage manager (includes advanced address space facilities support)
RMM	DFSMSrmm™
RRS	Resource recovery services (RRS)
SMF	System management facilities
SOM®	SOMobjects®
SMS	Storage management subsystem
SMSPDSE1	Optional restartable PDSE address space. If the SMSPDSE1 address space is started, SMSPDSE manages PDSEs in the LINKLST concatenatio and SMSPDSE1 manages all other PDSEs.
SMSVSAM	VSAM record level sharing
TCPIP	TCP/IP for MVS
TRACE	System trace
VLF	Virtual lookaside facility
XCFAS	Cross system coupling facility
VTAM®	VTAM
WLM	Workload management

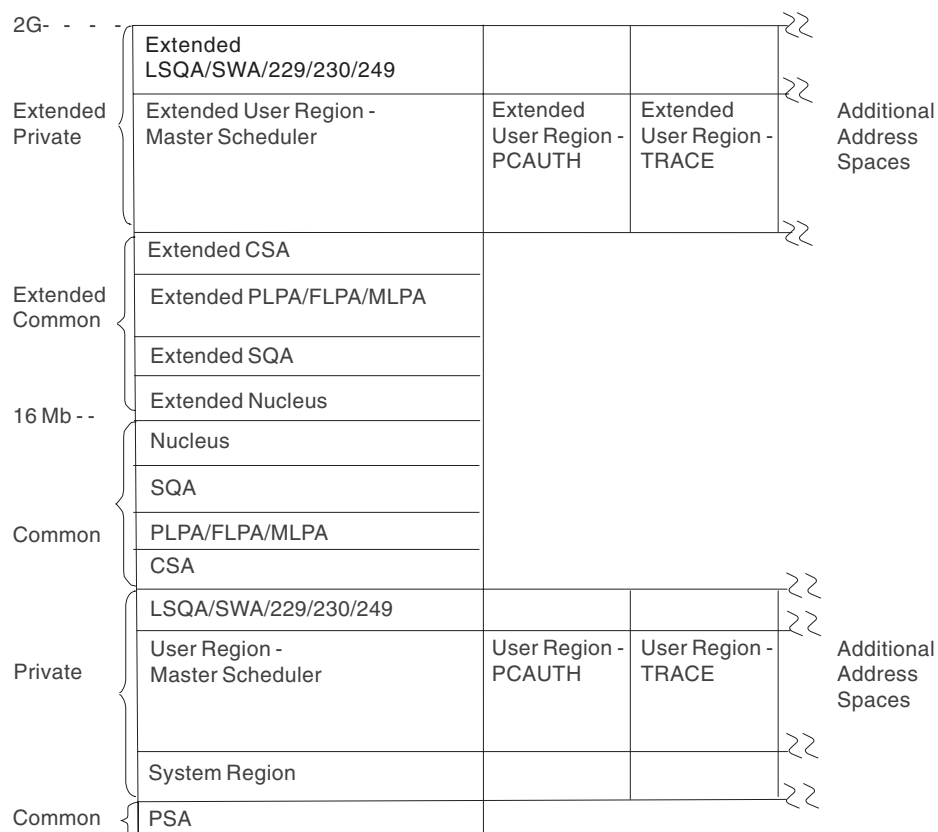


Figure 1-1. Virtual Storage Layout for Multiple Address Spaces

Address spaces differ in their ability to use system services depending on whether the address space is a limited function or full function address space.

- Limited function address space

If the specific initialization routines provided by the components that use IEEMB881 enter a wait state, pending Master Scheduler Initialization, STC does no additional address space initialization. Thus, the functions that component address spaces can perform are limited. Components with limited function address spaces cannot:

- Allocate data sets.
- Read JCL procedures from SYS1.PROCLIB.
- Allocate a SYSOUT file through the Job Entry Subsystem.
- Use some system services because the components frequently run in cross memory mode.

- Full function address space

If, after a component completes its own initialization, it returns to IEEPRWI2 and completes STC processing, the address space is fully initialized. Such an address space is called a full function address space.

The component creating a full function address space does not need to provide a procedure in SYS1.PROCLIB. If specified to IEEMB881, a common system address space procedure, IEESYSAS, will invoke a specified program to run in the address space.

For example, if a full function address space called FFA is to be started using the module IEEMB899, the component would ordinarily need to supply a procedure of the following form:

```
//FFA    PROC
//      EXEC PGM=IEEMB899
```

The procedure will be invoked as follows:

```
//IEESYSAS JOB
//FFA      EXEC IEESYSAS
```

The procedure IEESYSAS consists of the following statements:

```
//IEESYSAS PROC=IEFBR14
//      EXEC PGM=&PROG
```

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.

Note: When JES3 is the primary job entry subsystem, a second JES3 address space (JES3AUX) can be optionally initialized after master scheduler initialization completes. The JES3AUX address space is an auxiliary address space that contains JES3 control blocks and data.

Subsystem Initialization

Subsystem initialization is the process of readying a subsystem for use in the system. IEFSSNxx members of SYS1.PARMLIB contain the definitions for the primary subsystems, such as JES2 or JES3, and the secondary subsystems, such as VPSS and DB2®. For detailed information about the data contained in IEFSSNxx members for secondary systems, please refer to the installation manual for the specific system.

During system initialization, the defined subsystems are initialized. You should define the primary subsystem (JES) first, because other subsystems, such as DB2, require the services of the primary subsystem in their initialization routines. Problems can occur if subsystems that use the subsystem affinity service in their initialization routines are initialized before the primary subsystem. After the primary JES is initialized, then the subsystems are initialized in the order in which the IEFSSNxx parmlib members are specified by the SSN parameter. For example, for SSN=(aa,bb) parmlib member IEFSSNaa would be processed before IEFSSNbb.

Note: The storage management subsystem (SMS) is the only subsystem that can be defined before the primary subsystem. Refer to the description of parmlib member IEFSSNxx in *z/OS MVS Initialization and Tuning Reference* for SMS considerations.

Using IEFSSNxx to initialize the subsystems, you can specify the name of a subsystem initialization routine to be given control during master scheduler initialization, and you can specify the input parameter to be passed to the subsystem initialization routine. IEFSSNxx is described in more detail in *z/OS MVS Initialization and Tuning Reference*.

START/LOGON/MOUNT Processing

After the system is initialized and the job entry subsystem is active, jobs may 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 must be allocated. Note that before LOGON, the operator must have started TCAM or VTAM/TCAS, which have their own address spaces. Figure 1-1 is a virtual storage map

containing or naming the basic system component address spaces, the optional TCAM and VTAM system address spaces, and a user address space.

The system resources manager decides, based on resource availability, whether a new address space can be created. If not, the new address space will not be created until the system resources manager finds conditions suitable.

Processor Storage Overview

Processor storage only consists of real storage (formerly called central storage) in the z/Architecture mode. This section provides an overview of real storage. Note that unlike the combination of central and expanded storage in the ESA/390 environment, expanded storage is not supported in the z/Architecture mode.

The system uses a portion of both central storage and virtual storage. To determine how much central storage is available to the installation, the system's fixed storage requirements must be subtracted from the total central storage. The central storage available to an installation can be used for the concurrent execution of the paged-in portions of any installation programs.

The real storage manager (RSM) controls the allocation of central storage during initialization and pages in user or system functions for execution. Some RSM functions:

- Allocate central storage to satisfy GETMAIN requests for SQA and LSQA.
- Allocate central storage for page fixing.
- Allocate central storage for an address space that is to be swapped in.

If there is storage above 16 megabytes, RSM allocates central storage locations above 16 megabytes for SQA, LSQA, and the pageable requirements of the system. When non-fixed pages are fixed for the first time, RSM:

- Ensures that the pages occupy the appropriate type of frame
- Fixes the pages and records the type of frame used

Pages that must reside in central storage below 16 megabytes include:

- SQA subpool 226 pages.
- Fixed pages obtained using the RC, RU, VRC, or VRU form of GETMAIN if one of the following is true:
 - LOC=24 is specified.
 - LOC=RES, the default, is either specified or taken, and the program issuing the GETMAIN resides below 16 megabytes, runs in 24-bit mode, and has not requested storage from a subpool supported only above 16 megabytes.
- Fixed pages obtained using the LU, LC, EU, EC, VU, VC, or R form of GETMAIN.
- Storage whose virtual address and real address are the same (V=R pages).

Pages that can reside in central storage above 16 megabytes include:

- Nucleus pages.
- SQA subpools 239 and 245 pages.
- LSQA pages.
- All pages with virtual addresses greater than 16 megabytes.
- Fixed pages obtained using the RC, RU, VRC, or VRU form of GETMAIN if one of the following is true:
 - LOC=(24,31) is specified.

- LOC=(RES,31) is specified.
- LOC=31 is specified.
- LOC=(31,31) is specified.
- LOC=RES, the default, is either specified or taken, and the program issuing the GETMAIN resides above 16 megabytes virtual.
- LOC=RES, the default, is either specified or taken, and the program issuing the GETMAIN resides below 16 megabytes virtual, but runs in 31-bit mode and has requested storage from a subpool supported only above 16 megabytes.
- Any non-fixed page.

Note: The system backs nucleus pages in real storage below 2 gigabytes. You can however, back SQA and LSQA pages above 2 gigabytes when you specify LOC=(24,64) or LOC=(31,64).

Each installation is responsible for establishing many of the central storage parameters that govern RSM's processing. The following overview describes the function of each area composing central storage.

The primary requirements/areas composing central storage are:

1. The basic system fixed storage requirements — the nucleus, the allocated portion of SQA, and the fixed portion of CSA.
2. The private area fixed requirements of each swapped-in address space — the LSQA for each address space and the page-fixed portion of each virtual address space.

Once initialized, the basic system fixed requirements (sometimes called global system requirements) remain the same until system parameters are changed. Fixed storage requirements (or usage) will, however, increase as various batch or time sharing users are swapped-in. Thus, to calculate the approximate fixed storage requirements for an installation, the fixed requirements for each swapped-in address space must be added to the basic fixed system requirements. Fixed requirements for each virtual address space include system storage requirements for the LSQA (which is fixed when users are swapped in) and the central storage estimates for the page-fixed portions of the installation's programs.

The central storage for the processor, reduced by the global fixed and paged-in virtual storage required to support installation options, identifies the central storage remaining to support swapped-in address spaces. The total number of jobs that can be swapped in concurrently can be determined by estimating the working set (the amount of virtual storage that must be paged in for the program to run effectively) for each installation program. The working set requirements will vary from program to program and will also change dynamically during execution of the program. Allowances should be made for *maximum* requirements when making the estimates.

System Preferred Area

To enable a V=R allocation to occur and storage to be varied offline, MVS performs special handling for the following types of pages:

- SQA
- LSQA for non-swappable address spaces
- Fixed page frame assignments for non-swappable address spaces.

Because MVS cannot, upon demand, free the frames used for these page types, central storage could become fragmented (by the frames that could not be freed.) Such fragmentation could prevent a V=R allocation or prevent a storage unit from being varied offline. Therefore, for all storage requests for the types of pages noted, RSM allocates storage from the preferred area to prevent fragmentation of the nonpreferred “reconfigurable” area.

A system parameter, RSU, allows the installation to specify the number of storage units that are to be kept free of long-term fixed storage allocations, and thus be available for varying offline. Once this limit is established, the remainder of central storage, excluding storage reserved for V=R allocation, is marked as preferred area storage and used for long-term fixed storage allocation.

Nucleus Area

The nucleus area contains the nucleus load module and extensions to the nucleus that are initialized during IPL processing.

The nucleus includes a base and an architectural extension. Specify the correct architectural extension with the ARCHLVL statement in the LOADxx member of SYS1.PARMLIB for your system to run in either ESA/390 mode or z/Architecture mode.

The Fixed Link Pack Area (FLPA)

An installation can elect to have some modules that are normally loaded in the pageable link pack area (PLPA) loaded into the fixed link pack area (FLPA). This area should be used only for modules that significantly increase performance when they are fixed rather than pageable. Modules placed in the FLPA must be reentrant and refreshable.

The FLPA exists only for the duration of an IPL. Therefore, if an FLPA is desired, the modules in the FLPA must be specified for each IPL (including quick-start and warm-start IPLs).

It is the responsibility of the installation to determine which modules, if any, to place in the FLPA. Note that if a module is heavily used and is in the PLPA, the system's paging algorithms will tend to keep that module in central storage. The best candidates for the FLPA are modules that are infrequently used but are needed for fast response to some terminal-oriented action.

Specified by: A list of modules to be put in FLPA must be established by the installation in the fixed LPA list (IEAFIXxx) member of SYS1.PARMLIB. Modules from any partitioned data set can be included in the FLPA. FLPA is selected through specification of the FIX system parameter in IEASYSxx or from the operator's console at system initialization.

Any module in the FLPA will be treated by the system as though it came from an APF-authorized library. Ensure that you have properly protected any library named in IEAFIXxx to avoid system security and integrity exposures, just as you would protect any APF-authorized library. This area may be used to contain reenterable routines from either APF-authorized or non-APF-authorized libraries that are to be part of the pageable extension to the link pack area during the current IPL.

System Queue Area (SQA-Fixed)

SQA is allocated in fixed storage upon demand as long-term fixed storage and remains so until explicitly freed. The number of central frames assigned to SQA may increase and decrease to meet the demands of the system.

All SQA requirements are allocated in 4K frames as needed. These frames are placed within the preferred area (above 16 megabytes, if possible) to keep long-term resident pages grouped together.

If no space is available within the preferred area, and none can be obtained by stealing a non-fixed/unchanged page, then the “reconfigurable area” is reduced by one storage increment and the increment is marked as preferred area storage. An increment is the basic unit of physical storage. If there is no “reconfigurable area” to be reduced, a page is assigned from the V=R area. Excluded from page stealing are frames that have been fixed (for example, through the PGFIX macro), allocated to a V=R region, placed offline using a CONFIG command, have been changed, have I/O in progress, or contain a storage error.

Fixed LSQA Storage Requirements

Except for the extended private area page tables, which are pageable, the local system queue area (LSQA) for any swapped-in address space is fixed in central storage (above 16 megabytes, if possible). It remains so until it is explicitly freed or until the end of the job step or task associated with it. The number of LSQA frames allocated in central storage might increase or decrease to meet the demands of the system. If preferred storage is required for LSQA and no space is available in the preferred area, and none can be obtained by stealing a non-fixed/unchanged page, then the “reconfigurable area” is reduced by one storage increment, and the increment is marked as preferred area storage. If there is no “reconfigurable area” to be reduced, a page is assigned from the V=R area.

V=R Area

This area is used for the execution of job steps specified as fixed because they are assigned to V=R regions in virtual storage (see “Real Regions” on page 1-26). Such jobs run as nonpageable and nonswappable.

The V=R area is allocated starting directly above the system region in central storage. The virtual addresses for V=R regions are mapped one-to-one with the central addresses in this area. When a job requests a V=R region, the lowest available address in the V=R area in central storage, followed by a contiguous area equal in size to the V=R region in virtual storage, is located and allocated to the region.

If there is not enough V=R space available in the V=R area, the allocation and execution of new V=R regions are prohibited until enough contiguous storage is made available.

The V=R area can become fragmented because of system allocation for SQA and LSQA or because of long-term fixing. When this happens, it becomes more difficult — and may be impossible — for the system to find contiguous storage space for allocating V=R regions. Such fragmentation may last for the duration of an IPL. It is possible that fragmentation will have a cumulative effect as long-term fixed pages are occasionally assigned frames from the V=R area.

Specified by:

- The REAL parameter of the IEASYSxx member
- Use of the REAL parameter from the operator's console during NIP.

Expanded Storage Overview

Expanded storage information has been deleted because expanded storage is no longer supported in the z/Architecture mode.

Virtual Storage Overview

Estimating the virtual storage allocated at an installation is important primarily because this storage must be backed up by central storage in some ratio (for example, 25%). This backup storage contributes significantly to an installation's total central storage requirements.

Virtual storage must also be backed up by expanded storage or auxiliary storage. For information about estimating the amount of storage your installation will need, see the discussion of paging data space in Chapter 2, "Auxiliary Storage Management Initialization."

Each installation can use virtual storage parameters to specify how certain virtual storage areas are to be allocated. These parameters have an impact on central storage use and overall system performance. The following overview describes the function of each virtual storage area. For information about identifying problems with virtual storage requests, see "Identifying Problems in Virtual Storage (DIAGxx Parmlib Member)" on page 1-29.

The Virtual Storage Address Space

A two-gigabyte virtual storage address space is provided for:

- The master scheduler address space
- JES
- Other system component address spaces, such as allocation, system trace, system management facilities (SMF), and dumping services
- Each user (batch or TSO/E).

The system uses a portion of each virtual address space.

Each virtual address space consists of:

- The **common area** below 16 megabytes
- The **private area** below 16 megabytes
- The **extended common area** above 16 megabytes
- The **extended private area** above 16 megabytes.

Figure 1-2 on page 1-11 shows the layout of the storage areas for an address space in virtual storage. Note that most of the system areas exist both below and above 16 megabytes, providing an environment that can support 24-bit, 31-bit, and 64-bit addressing. However, each area and its counterpart above 16 megabytes can be thought of as a single logical area in virtual storage.

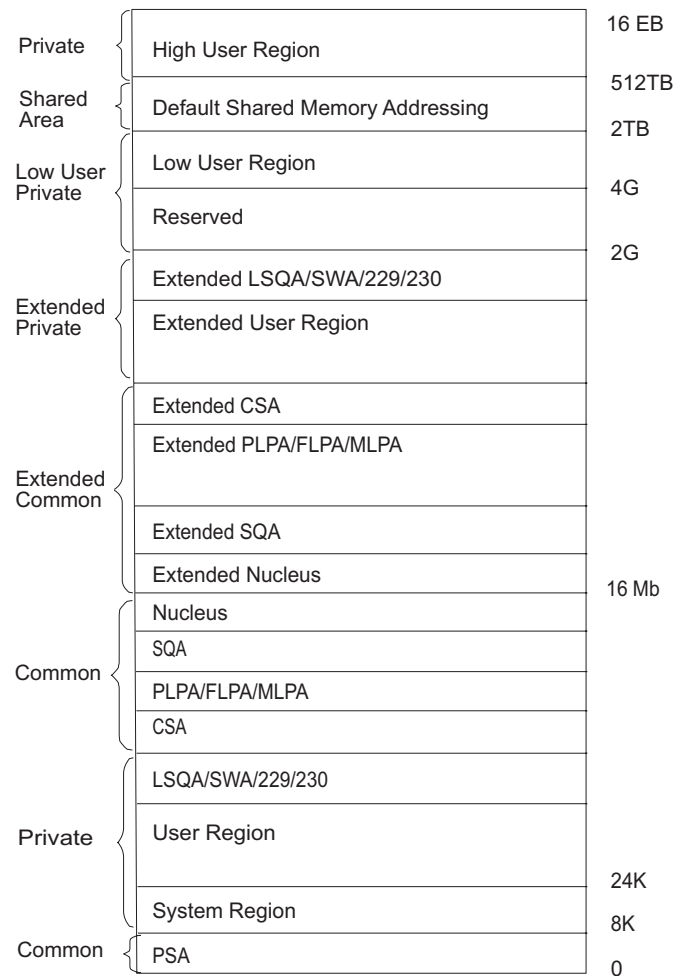


Figure 1-2. Virtual Storage Layout for a Single Address Space

The **common area** contains system control programs and control blocks. The following storage areas are located in the common area:

- Prefixed storage area (PSA).
- Common service area (CSA).
- Pageable link pack area (PLPA).
- Fixed link pack area (FLPA).
- Modified link pack area (MLPA).
- System queue area (SQA).
- Nucleus, which is fixed and nonswappable.

Each storage area in the common area (below 16 megabytes) has a counterpart in the extended common area (above 16 megabytes) with the exception of the PSA.

For more information about using storage above the 2-gigabyte address, see *z/OS MVS Programming: Extended Addressability Guide*.

Each address space uses the same common area. Portions of the common area are paged in and out as the demands of the system change and as new user jobs (batch or time-shared) start and old ones terminate.

The **private area** contains:

- A local system queue area (LSQA).

- A scheduler work area (SWA).
- Subpools 229, 230, and 249 (the authorized user key area).
- A 16K system region area.
- Either a V=V (virtual = virtual) or V=R (virtual = real) private user region for running programs and storing data.

Except for the 16K system region area and V=R user regions, each storage area in the private area below 16 megabytes has a counterpart in the extended private area above 16 megabytes.

Each address space has its own unique private area allocation. The private area (except LSQA) is pageable unless a user specifies a V=R region. If assigned as V=R, the actual V=R region area (excluding SWA, the 16K system region area, and subpools 229, 230, and 249) is fixed and nonswappable.

General Virtual Storage Allocation Considerations

Virtual storage allocated in each address space is divided between the system's requirements and the user's requirements. The base system control programs require space from each of the basic areas.

Storage for SQA, CSA, LSQA, and SWA is assigned for either the system or a specific user. Generally, space is assigned to the system in SQA and CSA and, for users, in LSQA or SWA.

System Queue Area (SQA/Extended SQA)

This area contains tables and queues relating to the entire system. Its contents are highly dependent on configuration and job requirements at an installation. The total amount of virtual storage and number of private virtual storage address spaces are two of the factors that affect the system's use of SQA.

The SQA is allocated directly below the nucleus; the extended SQA is allocated directly above the extended nucleus.

The size of the SQA can be specified through the:

- SQA parameter in the IEASYSxx member of SYS1.PARMLIB
- NIP or operator's console.

If the specified amount of virtual storage is not available during initialization, a warning message will be issued. The SQA parameter may be respecified at that time from the operator's console.

Virtual SQA is allocated as a number of 64K blocks to be added to the minimum system requirements for SQA. If the SQA required by the system configuration exceeds the amount that has been reserved through the SQA parameter, the system attempts to allocate additional virtual SQA from the CSA area. When certain storage thresholds are reached, as explained below, the system stops creating new address spaces. When SQA is in use, it is fixed in central storage.

The size of the SQA cannot be increased or decreased by the operator during a restart that reuses the previously initialized PLPA (a quick start). The size will be the same as during the preceding IPL.

SQA/CSA Thresholds

Ensuring the appropriate size of the extended SQA and extended CSA storage is critical to the long-term operation of the system.

If the size allocated for SQA is too large, the thresholds for the IRA100E and IRA101E messages will not be met and CSA can become exhausted and cause a system outage. One way to avoid this problem is to allocate the minimum SQA required or allow for some CSA conversion so that the storage thresholds that trigger the IRA100E and IRA101E messages are based only on the remaining CSA.

If the size allocated for extended SQA is too small or is used up very quickly, the system attempts to use extended CSA. When both extended SQA and extended CSA are used up, the system allocates space from SQA and CSA below 16 megabytes. The allocation of this storage could eventually lead to a system failure.

- When the size, in bytes, of combined total of free SQA + CSA pages falls below the "high insufficient" threshold, the system issues message IRA100E
- If the size, in bytes, of available SQA and SQA pages falls below the "low insufficient" threshold, the system issues message IRA101E
- If storage is freed such that the available SQA+CSA amount reaches the "high sufficient" threshold, the system issues message IRA102I

The following conditions may be responsible for a shortage of SQA/CSA:

- There has been storage growth beyond the previous normal range.
- Allocation of SQA and/or CSA is inadequate.
- The current thresholds at which the IRA100E and/or IRA101E messages are issued are too high for your installation.

Note: IBM® recommends that you do not change the storage thresholds set by the system. Setting the thresholds too low can hamper the ability of the system to recover from storage shortages and may result in unscheduled system outages. Setting the thresholds too high can cause IRA100E, IRA101E, and IRA102I messages to be issued excessively. If you do change the storage thresholds, you should be sure that the threshold is not being reached because of a problem in the system or inadequate allocation of CSA/SQA.

The SQA/CSA threshold levels are contained in the IGVDCCLIM CSECT in load module IEAIPLO4. The following are the offsets of the threshold values:

Table 1-1. Offsets for the SQA/CSA threshold levels

OFFSET	Default Value	System Enforced Minimum	Description	Comments
0000	x'41000'	x'9000'	Sufficient space high	IRA102I message issued at this threshold
0004	x'21000'	x'5000'	Sufficient space low	IRA102I message issued at this threshold
0008	x'40000'	x'8000'	Insufficient space high	IRA100E message issued at this threshold
000C	x'20000'	x'4000'	Insufficient space low	IRA101E message issued at this threshold

If you change the default thresholds, make sure that the new sufficient threshold values are at least x'1000' larger than the insufficient values. The new values become effective at the next IPL.

Pageable Link Pack Area (PLPA/Extended PLPA)

This area contains SVC routines, access methods, and other 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. Any module in the pageable link pack area will be treated by the system as though it came from an APF-authorized library. Ensure that you have properly protected SYS1.LPALIB and any library named in LPALSTxx or on an LPA statement in PROGxx to avoid system security and integrity exposures, just as you would protect any APF-authorized library.

It is desirable to place all frequently used refreshable SYS1.LINKLIB and SYS1.CMDLIB modules in the PLPA because of the following advantages:

- If possible, PLPA is backed by central storage above 16 megabytes; central storage below 16 megabytes is then available for other uses.
- The length of time that a page occupies central storage depends on its frequency of use. If the page is not used over a period of time, the system will reuse (steal) the central storage frame that the page occupies.
- The most frequently used PLPA modules in a time period will tend to remain in central storage.
- PLPA paged-in modules avoid program fetch overhead.
- Two or more programs that need the same PLPA module share the common PLPA code, thus reducing the demand for central storage.
- The main cost of unused PLPA modules is paging space, because only auxiliary storage is involved when modules are not being used.
- All modules in the PLPA are treated as refreshable, and are not paged-out. This action reduces the overall paging rate compared with modules in other libraries.

See “Placing Modules in the System’s Search Order for Programs” for an alternative suggestion on the placement of some PLPA and SYS1.CMDLIB modules. Any installation may also specify that some reenterable modules from the LNKLIB concatenation, SYS1.SVCLIB, and/or the LPALST concatenation be placed in a fixed extension to the link pack area (FLPA) to further improve performance (see “The Fixed Link Pack Area (FLPA)” on page 1-8).

Modules loaded into the PLPA are packed within page boundaries. Modules larger than 4K begin on a page boundary with smaller modules filling out. PLPA can be used more efficiently through use of the LPA packing list (IEAPAKxx). IEAPAKxx allows an installation to pack groups of related modules together, which can sharply reduce page faults. The total size of modules within a group should not exceed 4K, and the residence mode (RMODE) of the modules in a group should be the same. For more information about IEAPAKxx, see *z/OS MVS Initialization and Tuning Reference*.

Placing Modules in the System’s Search Order for Programs

Modules (programs), whether stored as load modules or program objects, must be loaded into both virtual storage and central storage before they can be run. When one module calls another module, either directly by asking for it to be run or indirectly by requesting a system service that uses it, it does not begin to run instantly. How long it takes before a requested module begins to run depends on where in its search order the system finds a usable copy and on how long it takes the system to make the copy it finds available.

You should consider these factors when deciding where to place individual modules or libraries containing multiple modules in the system-wide search order for modules:

- The search order the system uses for modules
- How placement affects virtual storage boundaries
- How placement affects system performance
- How placement affects application performance

The Search Order the System uses 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. Data sets concatenated to TASKLIB are searched for after JPA but before any specified STEPLIB or JOBLIB. 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. For more information about TASKLIB, see *z/OS MVS Programming: Assembler Services Guide*.

3. STEPLIB or JOBLIB

STEPLIB and JOBLIB 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. For more information about JOBLIB and STEPLIB, see *z/OS MVS JCL Reference*.

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 LNKLISTxx.

By default, the linklist begins with SYS1.LINKLIB, SYS1.MIGLIB, SYS1.CSSLIB, SYS1.SIEALNKE, and SYS1.SIEAMIGE. 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.

Note:

1. For more information about which system services load modules, see:
 - a. *z/OS MVS Programming: Assembler Services Guide*
 - b. *z/OS MVS Programming: Assembler Services Reference ABE-HSP*
 - c. *z/OS MVS Programming: Authorized Assembler Services Guide*
 - d. *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*
 - e. *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*
 - f. *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*
 - g. *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*
2. 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. For more information about these parameters, see the topic about the search for the load module in *z/OS MVS Programming: Assembler Services Guide*.
3. Some IBM subsystems (notably CICS® and IMS™) and applications (such as ISPF) use the facilities described in the above note to establish other search orders for programs.
4. A copy of a module already loaded in virtual storage might not be accessible to another module that needs it. For example, the copy might reside in another address space, or might have been used or be in use and not be reusable or reentrant. Whenever an accessible copy is not available, any module to be used must be loaded. For more information about the system's search order for programs and when modules are usable or unusable, see the chapter on Program Management in *z/OS MVS Programming: Assembler Services Guide*.

How Placement Affects Application Performance

Modules begin to run most quickly when all these conditions are true:

- They are already loaded in virtual storage
- The virtual storage they are loaded into is accessible to the programs that call them
- The copy that is loaded is usable
- The virtual storage is backed by central storage (that is, the virtual storage pages containing the programs are not paged out).

Modules that are accessible and usable (and have already been loaded into virtual storage but not backed in central storage) must be returned to central storage from either expanded storage or page data sets on DASD. Modules in the private area and those in LPA (other than in FLPA) can be in virtual storage without being backed by central storage. Because I/O is very slow compared to storage access, these modules will begin to run much faster when they are in central storage or in expanded storage than when they are not. Because data must be copied from expanded storage to central storage, modules can begin to run most quickly when they are in central storage.

Modules placed anywhere in LPA are always in virtual storage, and modules placed in FLPA are also always in central storage. Whether modules in LPA, but outside FLPA, are in central storage depends on how often they are used by all the users of the system, and on how much central storage is available. The more often an LPA module is used, and the more central storage is available on the system, the more likely it is that the pages containing the copy of the module will be in central storage at any given time.

LPA pages are only stolen, and never paged out, because there are copies of all LPA pages in the LPA page data set. But the results of paging out and page stealing are usually the same; unless stolen pages are reclaimed before being used for something else, they will not be in central storage when the module they contain is called.

LPA modules must be referenced very often 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. One reason these pages might be stolen is that address spaces often get swapped out (without the PLPA pages to which they refer), and a swapped-out address space cannot refer to a page in LPA.

When all the pages containing an LPA module (or its first page) are not in central storage when the module is called, the module will begin to run only after its first page has been brought into central storage.

Modules can also be loaded into CSA by authorized programs. When modules are loaded into CSA and shared by multiple address spaces, the performance considerations are similar to those for modules placed in LPA. (However, unlike LPA pages, CSA pages must be paged out when the system reclaims them.)

When a usable and accessible copy of a module cannot be found in virtual storage, either the request must be deferred or the module must be loaded. When the module must be loaded, it can be loaded from a VLF data space used by LLA, or from load libraries or PDSEs residing on DASD.

Modules not in LPA must always be loaded the first time they are used by an address space. How long this takes depends on:

- Whether the directory for the library in which the module resides is cached
- Whether the module itself is cached in storage
- The response time of the DASD subsystem on which the module resides at the time the I/O loads the module.

The LLA address space caches directory entries for all the modules in the data sets in the linklist concatenation (defined in PROGxx and LNKLISTxx) by default. Because the directory entries are cached, the system does not need to read the data set directory to find out where the module is before fetching it. This reduces I/O significantly. In addition, unless the system defaults are changed, LLA will use VLF to cache small, frequently-used load modules from the linklist. A module cached in VLF by LLA can be copied into its caller's virtual storage much more quickly than the module can be fetched from DASD.

You can control the amount of storage used by VLF by specifying the MAXVIRT parameter in a COFVLFxx member of PARMLIB. You can also define additional libraries to be managed by LLA and VLF. For more information about controlling VLF's use of storage and defining additional libraries, see *z/OS MVS Initialization and Tuning Reference*.

When a module is called, no accessible or usable copy of it exists in central or expanded storage, and it is not cached by LLA, the system must bring it in from DASD. Unless the directory entry for the module is cached, this involves at least two sets of I/O operations. The first reads the data set's directory to find out where the module is stored, and the second reads the member of the data set to load the module. The second I/O operation might be followed by additional I/O operations to finish loading the module when the module is large or when the system, channel subsystem, or DASD subsystem is heavily loaded.

How long it takes to complete these I/O operations depends on how busy all of the resources needed to complete them are. These resources include:

- The DASD volume
- The DASD controller
- The DASD control unit
- The channel path
- The channel subsystem
- The CPs enabled for I/O in the processor
- The number of SAPs (CMOS processors only).

In addition, if cached controllers are used, the reference patterns of the data on DASD will determine whether a module being fetched will be in the cache. Reading data from cache is much faster than reading it from the DASD volume itself. If the fetch time for the modules in a data set is important, you should try to place it on a volume, string, control unit, and channel path that are busy a small percentage of the time, and behind a cache controller with a high ratio of cache reads to DASD reads.

Finally, the time it takes to read a module from a load library (not a PDSE) on DASD is minimized when the modules are written to a data set by the binder, linkage editor, or an IEBCOPY COPYMOD operation when the data set has a block size equal to or greater than the size of the largest load module or, if the library contains load modules larger than 32 kilobytes, set to the maximum supported block size of 32760 bytes.

Access Time for Modules: From a performance standpoint, modules not already loaded in an address space will usually be available to a program in the least time when found at the beginning of the list below, and will take more time to be available when found later in the list. Remember that the system stops searching for a module once it has been found in the search order; so, if it is present in more than one place, only the first copy found will be used. The placement of the first copy in the search order will affect how long it takes the system to make the module available. Possible places are:

1. LPA
2. Link list concatenation (all directory entries and some modules cached automatically)
3. TASKLIB/STEPLIB/JOBLIB (with LLA caching of the library)
4. TASKLIB/STEPLIB/JOBLIB (without LLA caching of the library).

For best application performance, you should place as many frequently-used modules as high on this list as you can. However, the following system-wide factors must be considered when you decide how many load modules to place in LPA:

- Performance
When central storage is not constrained, frequently-used LPA routines almost always reside in central storage, and access to these modules will be very fast.
- Virtual Storage

How much virtual storage is available for address spaces that use the modules placed in LPA, and how much is available for address spaces that do not use the modules placed in LPA.

How Placement Affects System Performance

Whether the placement of a module affects system performance depends on how many address spaces use the module and on how often the module is used. Placement of infrequently-used modules that are used by few address spaces have little effect on system-wide performance or on the performance of address spaces that do not use the modules. Placement of frequently-used modules used by a large number of address spaces, particularly those used by a number of address spaces at the same time, can substantially affect system performance.

Placement of Modules in LPA: More central storage can be used when a large number of address spaces each load their own copy of a frequently-used module, because multiple copies are more likely to exist in central storage at any time. One possible consequence of increased central storage use is increased use of expanded storage or increased paging.

When frequently-used modules are placed in LPA, all address spaces share the same copy, and central storage usage tends to be reduced. The probability of reducing central storage usage increases with the number of address spaces using a module placed in LPA, with the number of those address spaces usually swapped in at any given time, and with how often the address spaces reference the module. You should consider placing in LPA modules used very often by a large number of address spaces.

By contrast, if few address spaces load a module, it is less likely that multiple copies of it will exist in central storage at any one time. The same is true if many address spaces load a module, run it once, and then never run it again, as might happen for those used only when initializing a function or an application. This is also true when many address spaces load a module but use it infrequently, even when a large number of these address spaces are often swapped in at one time; for example, some modules are used only when unusual circumstances arise within an application. Modules that fit these descriptions are seldom good candidates for placement in LPA.

You can add modules to LPA in these ways:

- Add the library containing the modules to the LPA list
Any library containing only reentrant modules can be added to the LPA list, which places all its modules in LPA. Note that modules are added to LPA from the first place in the LPA list concatenation they are found.
- Add the modules to dynamic LPA
Use this approach instead of placing modules in FLPA or MLPA whenever possible. Searches for modules in dynamic LPA are approximately as fast as those for modules in LPA, and placement of modules in dynamic LPA imposes no overhead on searches for other modules. Note that another LPA module whose address was stored before a module with the same name was loaded into dynamic LPA might continue to be used.
- Add the modules to FLPA
Modules in the fixed LPA list will be found very quickly, but at the expense of modules that must be found in the LPA directory. It is undesirable to make this list very long because searches for other modules will be prolonged. Note that modules placed in IEAFIXxx that reside in an LPA List data set will be placed in LPA twice, once in PLPA and once in FLPA.

- Add the modules to MLPA

Like placement in FLPA, placing a large number of modules in MLPA causes searches for all modules to be delayed, and this should be avoided. The delay will be proportional to the number of modules placed in MLPA, and it can become significant if you place a large number of modules in MLPA.

Placement of Modules Outside LPA: In addition to the effects on central storage, channel subsystem and DASD subsystem load are increased when a module is fetched frequently from DASD. How much it increases depends on how many I/O operations are required to fetch the module and on the size of the module to be fetched.

How Placement Affects Virtual Storage

When a module is loaded into the private area for an address space, the region available for other things is reduced by the amount of storage used for the module. Modules loaded from anywhere other than LPA (FLPA, MLPA, dynamic LPA, or PLPA) will be loaded into individual address spaces or into CSA.

When a module is added to LPA below 16 megabytes, the size of the explicitly-allocated common area below 16 megabytes will be increased by the amount of storage used for the module. When the explicitly-allocated common area does not end on a segment boundary, IPL processing allocates additional CSA down to the next segment boundary. Therefore, which virtual storage boundaries change when modules are added to LPA depends on whether a segment boundary is crossed or not.

When modules are added to LPA below 16 megabytes, and this does **not** result in the expansion of explicitly-allocated common storage past a segment boundary, less virtual storage will be available for CSA (and SQA overflow) storage. The amounts of CSA and SQA specified in IEASYSxx will still be available, but the system will add less CSA to that specified during IPL.

When the addition of modules to LPA does not result in a reduction in the size of the private area below 16 megabytes, adding load modules to LPA increases the amount of private area available for address spaces that use those load modules. This is because the system uses the copy of the load module in LPA rather than loading copies into each address space using the load module. In this case, there is no change to the private area storage available to address spaces that do not use those load modules.

When modules are added to LPA below 16 megabytes, the growth in LPA can cause the common area below 16 megabytes to cross one or more segment boundaries, which will reduce the available private area below 16 megabytes by a corresponding amount; each time the common area crosses a segment boundary, the private area is reduced by the size of one segment. The segment size in OS/390 is one megabyte.

When the size of the private area is reduced as a result of placing modules in LPA below 16 megabytes, the private area virtual storage available to address spaces that use these modules might or might not be changed. For example, if an address space uses 1.5 megabyte of modules, all of them are placed in LPA below 16 megabytes, and this causes the common area to expand across two segment boundaries, .5 megabytes less private area storage will be available for programs in that address space. But if adding the same 1.5 megabytes of modules causes only one segment boundary to be crossed, .5 megabytes more will be available, and adding exactly 1 megabytes of modules would cause no change in the amount of

private area storage available to programs in that address space. (These examples assume that no other changes are made to other common virtual storage area allocations at the same time.)

When the size of the private area is reduced as a result of placing modules in LPA below 16 megabytes, less storage will be available to all address spaces that do **not** use those modules.

A process similar to the process described for LPA is used when ELPA, the other Extended common areas, and the Extended private area are built above 16 megabytes. The only difference is that common storage areas above 16 megabytes are built from 16 megabytes upward, while those below 16 megabytes are built from 16 megabytes downward.

Modules can also be loaded in CSA, and some subsystems (like IMS) make use of this facility to make programs available to multiple address spaces. The virtual storage considerations for these modules are similar to those for LPA.

Recommendations for Improving System Performance

The following recommendations should improve system performance. They assume that the system's default search order will be used to find modules. You should determine what search order will be used for programs running in each of your applications and modify these recommendations as appropriate when other search orders will be used to find modules.

- Determine how much private area, CSA, and SQA virtual storage are required to run your applications, both above 16 megabytes and below.
- Determine which modules or libraries are important to the applications you care most about. From this list, determine how many are reentrant to see which are able to be placed in LPA. Of the remaining candidates, determine which can be safely placed in LPA, considering security and system integrity.

Note: All modules placed in LPA are assumed to be authorized. IBM publications identify libraries that can be placed in the LPA list safely, and many list modules you should consider placing in LPA to improve the performance of specific subsystems and applications.

Note that the system will try to load RMODE(ANY) modules above 16 megabytes whenever possible. RMODE(24) modules will always be loaded below 16 megabytes.

- To the extent possible without compromising required virtual storage, security, or integrity, place libraries containing a high percentage of frequently-used reentrant modules (and containing no modules that are not reentrant) in the LPA list. For example, if TSO/E response time is important and virtual storage considerations allow it, add the CMDLIB data set to the LPA list.
- To the extent possible without compromising available virtual storage, place frequently or moderately-used refreshable modules from other libraries (like the linklist concatenation) in LPA using dynamic LPA or MLPA. Make sure you do not inadvertently duplicate modules, module names, or aliases that already exist in LPA. For example, if TSO/E performance is important, but virtual storage considerations do not allow CMDLIB to be placed in the LPA list, place only the most frequently-used TSO/E modules on your system in dynamic LPA.

Use dynamic LPA to do this rather than MLPA whenever possible. Modules that might be used by the system before a SET PROG command can be processed cannot be placed solely in dynamic LPA. If these modules are not required in LPA before a SET PROG command can be processed, the library in which they reside can be placed in the linklist so they are available before a SET PROG can

be processed, but enjoy the performance advantages of LPA residency later. For example, Language Environment® runtime modules required by z/OS UNIX System Services initialization can be made available by placing the SCEERUN library in the linklist, and performance for applications using Language Environment (including z/OS UNIX System Services) can be improved by also placing selected modules from SCEERUN in dynamic LPA.

For more information about dynamic LPA, see the section on PROGxx in *z/OS MVS Initialization and Tuning Reference*. For information about MLPA, see the section on IEALPAXx in the same book in *z/OS MVS Initialization and Tuning Reference*.

To load modules in dynamic LPA, list them on an LPA ADD statement in a PROGxx member of PARMLIB. You can add or remove modules from dynamic LPA without an IPL using SET PROG=xx and SETPROG LPA operator commands. For more information, *z/OS MVS Initialization and Tuning Reference* and *z/OS MVS System Commands*.

- By contrast, do not place in LPA infrequently-used modules, those not important to critical applications (such as TSO/E command processors on a system where TSO/E response time is not important), and low-use user programs when this placement would negatively affect critical applications. Virtual storage is a finite resource, and placement of modules in LPA should be prioritized when necessary. Leaving low-use modules from the linklist (such as those in CMDLIB on systems where TSO/E performance is not critical) and low-use application modules outside LPA so they are loaded into user subpools will affect the performance of address spaces that use them and cause them to be swapped in and out with those address spaces. However, this placement usually has little or no effect on other address spaces that do not use these modules.
- Configure as much storage as possible as central storage. Configure only enough expanded storage to support workloads that rely specifically on the availability of ES plus a reasonable additional amount to provide a margin of safety. You can think of the use of ES for paging as if it were very fast I/O. When ES is used for paging, data must be copied from central storage to ES and back again. Before a page in ES can be migrated to auxiliary storage, it must first be copied back into central storage.
- If other measures (like WLM policy changes, managing the content of LPA, and balancing central and expanded storage allocations) fail to control storage saturation, and paging and swapping begin to affect your critical workloads, the most effective way to fix the problem is to add storage to the system. Sometimes, this is as simple as changing the storage allocated to different LPARs on the same processor. You should consider other options only when you cannot add storage to the system.
- If you experience significant PLPA paging, you can use the fixed LPA to reduce page-fault overhead and increase performance at the expense of central storage. You can assure that specific modules are kept in central storage by adding them to the fixed LPA by listing them in IEAFIXxx. This trade-off can be desirable with a system that tends to be CPU-bound, where it can be best to divert some of the central storage from possible use by additional address spaces, and use it for additional LPA modules.

High-usage PLPA modules probably need not be listed in IEAFIXxx because they tend to be referenced frequently enough to remain in central storage. A large FLPA makes less central storage available for pageable programs. Accordingly, fewer address spaces might be in central storage than would otherwise be the case. No loss in throughput should occur, however, if CPU use remains reasonably high.

Note that a large FLPA can increase other demand paging and swapping activity, and that this will impede the system's normal self-tuning actions because keeping these modules in storage might prevent other, more frequently-used modules, from being in storage as workloads shift over the course of time. Also, like module packing lists, fixed LPA lists need to be maintained when installing new releases of software, installing significant amounts of service, or when your workloads change. If you can prevent LPA paging by adding central storage, the system will be simpler to manage.

- When there is significant page-in activity for PLPA pages, and you cannot take other actions to reduce it economically, you can minimize page faults and disk arm movement by specifying module packing through the IEAPAKxx member of parmlib. Module packing reduces page faults by placing in the same virtual page those small modules (less than 4K bytes) that refer to each other frequently. In addition, module groups that frequently refer to each other but that exceed 4K bytes in combined size can be placed in adjacent (4K) auxiliary storage slots to reduce seek time. Thus, use of IEAPAKxx should improve performance compared with the simple loading of the PLPA from the LPALST concatenation. (See the description of parmlib member IEAPAKxx in *z/OS MVS Initialization and Tuning Reference*.)

However, you must maintain module packing lists whenever you install new levels of software or significant service, or when your workloads change. If you can increase the amount of central storage enough to control PLPA paging rather than using a module packing list, the system will be simpler to manage.

- If the first PLPA page data set specified during IPL is large enough, all PLPA pages are written to the same data set. If the first page data set is not large enough to contain all PLPA pages (for example, when allocated as a one-cylinder data set as recommended below), the remaining pages are written to the common page data set (the second one specified during IPL). For best performance, all PLPA pages would be written to a single page data set on a single DASD volume.

However, a reasonable alternative is to define the PLPA page data set as a single cylinder and define enough storage for the common page data set to contain both the PLPA and common pages. When defined this way, the PLPA and common page data sets should be contiguous, with the small PLPA data set followed immediately by the large common data set on the volume. You should consider allocating these data sets this way unless you experience significant PLPA paging, because it reduces the number of page data sets for which space must be managed and simplifies support.

- If you have significant swapping or paging activity that affects critical applications, and you cannot add storage to manage it, you can tune the paging subsystem.

When most paging subsystem activity is for swapping, a large number of page data sets can outperform a small number of page data sets, even on high-speed or cached devices. If you have substantial swapping, consider using eight or more page data sets on different low-use volumes on low-use control units and channel paths. However, these should generally be considered stop-gap solutions. If the storage demand continues to grow, tuning the paging subsystem will usually delay the inevitable for only a short time. In the long run, adding central and expanded storage is always a better solution.

Note: Some cached devices also do not support cached paging.

Modified Link Pack Area (MLPA/Extended MLPA)

This area may be used to contain reenterable routines from either APF-authorized or non-APF-authorized libraries that are to be part of the pageable extension to the

link pack area during the current IPL. Any module in the modified link pack area will be treated by the system as though it came from an APF-authorized library. Ensure that you have properly protected any library named in IEALPAxx to avoid system security and integrity exposures, just as you would protect any APF-authorized library.

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).

The MLPA is allocated just below the FLPA (or the PLPA, if there is no FLPA); the extended MLPA is allocated above the extended FLPA (or the extended PLPA if there is no extended FLPA). When the system searches for a routine, the MLPA is searched before the PLPA.

Note: Loading a large number of modules in the MLPA can increase fetch time for modules that are not loaded in the LPA. This could affect system performance.

The MLPA can be used at IPL time to temporarily modify or update the PLPA with new or replacement modules. No actual modification is made to the quick start PLPA stored in the system's paging data sets. The MLPA is read-only, unless NOPROT is specified on the MLPA system parameter.

Specified by:

- Including a module list as an IEALPAxx member of SYS1.PARMLIB; where xx is the specific list.
- Including the MLPA system parameter in IEASYSxx or specifying MLPA from the operator's console during system initialization.

Common Service Area (CSA/Extended CSA)

This area contains pageable and fixed data areas that are addressable by all active virtual storage address spaces. CSA normally contains data referenced by a number of system address spaces, enabling address spaces to communicate by referencing the same piece of CSA data.

CSA is allocated directly below the MLPA; extended CSA is allocated directly above the extended MLPA. If the virtual SQA space is depleted, the system will allocate additional SQA space from the CSA.

Specified by:

- The SYSP parameter at the operator's console to specify an alternative system parameter list (IEASYSxx) that contains a CSA specification.
- The CSA parameter at the operator's console during system initialization. This value overrides the current system parameter value for CSA that was established by IEASYS00 or IEASYSxx.

Note: If the size allocated for extended SQA is too small or is used up very quickly, the system attempts to steal space from extended CSA. When both extended SQA and extended CSA are used up, the system allocates space from SQA and CSA below 16 megabytes. The allocation of this storage could eventually lead to a system failure. Ensuring the appropriate size of extended SQA and extended CSA storage is critical to the long-term operation of the system.

SQA/CSA Shortage Thresholds

Ensuring the appropriate size of extended SQA and extended CSA storage is critical to the long-term operation of the system. If the size allocated for extended SQA is too small or is used up very quickly, the system attempts to use extended CSA. When both extended SQA and extended CSA are used up, the system allocates space from SQA and CSA below 16 megabytes. The allocation of this storage could eventually lead to a system failure.

- When the size, in bytes, of combined total of free SQA + CSA pages falls below the "high insufficient" threshold, the system issues message IRA100E
- If the size, in bytes, of available SQA and SQA pages falls below the "low insufficient" threshold, the system issues message IRA101E

For more information about SQA shortages and the thresholds, see "SQA/CSA Thresholds" on page 1-13.

Local System Queue Area (LSQA/Extended LSQA)

Each virtual address space has an LSQA. The area contains tables and queues associated with the user's address space.

LSQA is intermixed with SWA and subpools 229, 230, and 249 downward from the bottom of the CSA into the unallocated portion of the private area, as needed. Extended LSQA is intermixed with SWA and subpools 229, 230, and 249 downward from 2 gigabytes into the unallocated portion of the extended private area, as needed. (See Figure 1-2 on page 1-11.) LSQA will not be taken from space below the top of the highest storage currently allocated to the private area user region. Any job will abnormally terminate unless there is enough space for allocating LSQA.

Scheduler Work Area (SWA/Extended SWA)

This area contains control blocks that exist from task initiation to task termination. It includes control blocks and tables created during JCL interpretation and used by the initiator during job step scheduling. Each initiator has its own SWA within the user's private area.

To enable recovery, the SWA can be recorded on direct access storage in the JOB JOURNAL. SWA is allocated at the top of each private area intermixed with LSQA and subpools 229, 230, and 249.

Subpools 229, 230, 249 - Extended 229, 230, 249

This area allows local storage within a virtual address space to be obtained in the requestor's storage protect key. The area is used for control blocks that can be obtained only by authorized programs having appropriate storage protect keys. These control blocks were placed in storage by system components on behalf of the user.

These subpools are intermixed with LSQA and SWA. Subpool 229 is fetch protected; subpools 230 and 249 are not. All three subpools are pageable. Subpools 229 and 230 are freed automatically at task termination; subpool 249 is freed automatically at jobstep task termination.

System Region

This area is reserved for GETMAINS by all system functions (for example, ERPs) running under the region control tasks. It comprises 16K (except in the master scheduler address space, in which it has a 200K maximum) of each private area

immediately above the PSA. V=V region space allocated to user jobs is allocated upwards from the top of this area. This area is pageable and exists for the life of each address space.

The Private Area User Region/Extended Private Area User Region

The portion of the user's private area within each virtual address space that is available to the user's problem programs is called the **user region**.

Types of User Regions

There are two types of user regions: virtual (or V=V) and real (or V=R). Virtual and real regions are mutually exclusive; private areas can be assigned to V=R or V=V, but not to both. It is the installation's responsibility to determine the type of region to which jobs are assigned. Usually, V=R should be assigned to regions containing jobs that cannot run in the V=V environment, or that are not readily adaptable to it. Programs that require a one-to-one mapping from virtual to central storage, such as program control interruption (PCI) driven channel programs, are candidates for real regions.

Two significant differences between virtual and real regions are:

- How they affect an installation's central storage requirements
- How their virtual storage addresses relate to their central storage addresses.

For virtual regions, which are pageable and swappable, the system allocates only as many central storage frames as are needed to store the paged-in portion of the job (plus its LSQA). The processor translates the virtual addresses of programs running in virtual regions to locate their central storage equivalent.

For real regions, which are nonpageable and nonswappable, the system allocates and fixes as many central storage frames as are needed to contain the entire user region. The virtual addresses for real regions map one-for-one with central storage addresses.

Virtual Regions: Virtual regions begin at the top of the system region (see Figure 1-2) and are allocated upward through the user region to the bottom of the area containing the LSQA, SWA, and the user key area (subpools 229, 230, and 249). Virtual regions are allocated above 16 megabytes also, beginning at the top of the extended CSA, and upward to the bottom of the extended LSQA, SWA, and the user key area (subpools 229, 230, and 249).

As a portion of any V=V job is paged in, 4K multiples (each 4K multiple being one page) of central storage are allocated from the available central storage. Central storage is dynamically assigned and freed on a demand basis as the job executes. V=V region requests that specify a specific region start address, are supported only for restart requests, and must specify an explicit size through JCL (see "Specifying Region Size" on page 1-27).

Real Regions: Real regions begin at the top of the system region (see Figure 1-2) and are allocated upward to the bottom of the area containing LSQA, SWA, and the user key area (subpools 229, 230, and 249). Unlike virtual regions, real regions are only allocated below 16 megabytes.

The system assigns real regions to a virtual space within the private area that maps one-for-one with the real addresses in central storage below 16 megabytes. Central storage for the entire region is allocated and fixed when the region is first created.

Specifying Region Type: A user can assign jobs (or job steps) to virtual or real regions by coding a value of VIRT or REAL on the ADDRSPC parameter on the job's JOB or EXEC statement. For more information on coding the ADDRSPC parameter, see *z/OS MVS JCL Reference*.

The system uses ADDRSPC with the REGION parameter. The relationship between the ADDRSPC and REGION parameter is explained in *z/OS MVS JCL User's Guide*.

Region Size and Region Limit

What is Region Size?: The region size is the amount of storage in the user region available to the job, started task, or TSO/E user. The system uses region size to determine the amount of storage that can be allocated to a job or step when a request is made using the STORAGE or GETMAIN macros and a variable length is requested. The region size minus the amount of storage currently allocated, determines the maximum amount of storage that can be allocated to a job by any single variable-length GETMAIN request.

What is Region Limit?: The region limit is the maximum total storage that can be allocated to a job by any combination of requests for additional storage using the GETMAIN or STORAGE macros. It is, in effect, a second limit on the size of the user's private area, imposed when the region size is exceeded.

Specifying Region Size: Users can specify a job's region size by coding the REGION parameter on the JOB or EXEC statement. The system rounds all region sizes to a 4K multiple.

The region size value should be less than the region limit value to protect against programs that issue variable length GETMAINs with very large maximums, and then do not immediately free part of that space or free such a small amount that a later GETMAIN (possibly issued by a system service) causes the job to fail.

For V=V jobs, the region size can be as large as the entire private area, minus the size of LSQA/SWA/user key area (subpools 229, 230, and 249) and the system region (see Figure 1-2).

For V=R jobs, the REGION parameter value cannot be greater than the value of the REAL system parameter specified at IPL. If the user does not explicitly specify a V=R job's region size in the job's JCL, the system uses the VRREGN system parameter value in the IEASYS00 member of SYS1.PARMLIB.

For more information on coding the REGION parameter, see *z/OS MVS JCL Reference*.

Note: VRREGN should not be confused with the REAL system parameter. REAL specifies the total amount of central storage that is to be reserved for running all active V=R regions. VRREGN specifies the default subset of that total space that is required for an individual job that does not have a region size specified in its JCL.

An installation can override the VRREGN default value in IEASYS00 by:

- Using an alternate system parameter list (IEASYSxx) that contains the desired VRREGN parameter value.

- Specifying the desired VRREGN value at the operator's console during system initialization. This value overrides the value for VRREGN that was specified in IEASYS00 or IEASYSxx.

For V=R requests, if contiguous storage of at least the size of the REGION parameter or the system default is not available in virtual or central storage, a request for a V=R region is placed on a wait queue until space becomes available.

Note that V=R regions must be mapped one for one into central storage. Therefore, they do not have their entire virtual storage private area at their disposal; they can use only that portion of their private area having addresses that correspond to the contiguous central storage area assigned to their region, and to LSQA, SWA, and subpools 229, 230, and 249.

Limiting User Region Size

Why Control Region Size or Region Limit?: It is the installation's choice whether to control a program's maximum region size or region limit. The region size allowed to users can affect performance of the entire system. When there is no limit on region size and the system uses its default values, users might obtain so much space within a region (by repeated requests for small amounts of storage or a single request for a large amount) that no space would remain in the private area for the system to use. This situation is likely to occur when a program issues a request for storage and specifies a variable length with such a large maximum value that most or all of the space remaining in the private area is allocated to the request. If this program actively uses this large amount of space (to write tables, for example), it can affect central storage (also known as real storage) and thus impact performance.

To avoid an unexpected out-of-space condition, the installation should require the specification of some region size on the REGION parameter of JOB or EXEC JCL statements. Also, the installation can set system-wide defaults for region size through the job entry subsystem (JES) or with MVS installation exit routines. The installation can control user region limits for varying circumstances by selecting values and associating them with job classes or accounting information.

Using JES to Limit Region Size: After interpreting the user-coded JCL, JES will pass to MVS either the value requested by the user on the REGION parameter, or the installation-defined JES defaults. JES determines the REGION value based on various factors, including the source of the job, or the class of the job, or both.

The limit for the user region size below 16 megabytes equals (1) the region size that is specified in the JCL, plus 64K, or (2) the JES default, plus 64K, if no region size is specified in the JCL. The IBM default limit for the user region size above 16 megabytes is 32 megabytes.

Note: If the region size specified in the JCL is zero, the region will be limited by the size of the private area.

For more information on JES defaults, see either *z/OS JES2 Initialization and Tuning Reference* or *z/OS JES3 Initialization and Tuning Reference*, depending on which JES your system is using.

Using Exit Routines to Limit Region Size: Two MVS installation exits are available to override the region values that JES passes to MVS. The exit routines are called IEFUSI and IEALIMIT.

The installation can use either IEFUSI or IEALIMIT to change the job's limit for the region size below 16 megabytes of virtual storage. However, to change the limit for the region size above 16 megabytes, the installation must code the IEFUSI installation exit.

If your installation does not supply an IEFUSI exit routine, the system uses the default values in the IBM-supplied IEALIMIT exit routine to determine region size. To determine region limit, the system adds 64K to the default region size.

Region and MEMLIMIT values and limits set by the IEFUSI exit will not be honored for programs with the NOHONORIEFUSIREGION program property table (PPT) attribute specified. The NOHONORIEFUSIREGION PPT attribute can be specified in the SCHEDxx member of SYS1.PARMLIB, or as an IBM supplied PPT default. This PPT attribute is used to bypass IEFUSI region controls for specific programs that require a larger than normal region in order to successfully execute.

For more information on IEFUSI and IEALIMIT, see *z/OS MVS Installation Exits*.

Identifying Problems in Virtual Storage (DIAGxx Parmlib Member)

This section describes functions you can use to identify problems with virtual storage requests (such as excessive use of common storage, or storage that is not freed at the end of a job or address space). The functions are:

- Common storage tracking
- GETMAIN/FREEMAIN/STORAGE (GFS) trace.

You can control the status of these functions in the DIAGxx parmlib member.

Using the Common Storage Tracking Function

Common storage tracking collects data about requests to obtain or free storage in CSA, ECSA, SQA, and ESQA. You can use this data to identify jobs or address spaces that use up an excessive amount of common storage or have ended without freeing common storage. If those jobs or address spaces have code to free that storage when they are canceled, you might relieve the shortage and avoid an IPL if you cancel those jobs or address spaces using an operator command.

You can use Resource Measurement Facility (RMF™) or a compatible monitor program to display the data that the storage tracking function collects. You can also format storage tracking data in a dump using interactive problem control system (IPCS). For information on how to use IPCS to format common storage tracking data, see the description of the VERBEXIT VSMDATA subcommand in *z/OS MVS IPCS Commands*.

The OWNER parameter on the CPOOL BUILD, GETMAIN, and STORAGE OBTAIN macros assigns ownership of the obtained CSA, ECSA, SQA, or ESQA storage to a particular address space or the system. To get the most value from the common storage tracking function, ensure that authorized programs specify the OWNER parameter on all CPOOL BUILD, GETMAIN, and STORAGE OBTAIN macros that:

- Request storage in CSA, ECSA, SQA, or ESQA, *and*
- Have an owning address space that is *not* the home address space.

IBM recommends that common storage tracking always be activated.

You can turn storage tracking on by activating a DIAGxx parmlib member, either at IPL (specify DIAG=xx as a system parameter) or during normal processing (enter a SET DIAG=xx command). For more information about using SET DIAG=xx, see *z/OS MVS System Commands*.

If you do not specify a DIAGxx parmlib member at IPL, the system processes the default member DIAG00. If DIAG00 does not exist, common storage tracking will not be turned on. Common storage remains active until turned off through a SET DIAG=xx command. DIAG00 also turns off the GFS trace function, which is described in “Using GETMAIN/FREEMAIN/STORAGE (GFS) Trace.”

IBM also provides the DIAG01 parmlib member, which turns the common storage tracking function on, and DIAG02, which turns the common storage tracking function off. Your installation must create any additional DIAGxx parmlib members.

Using GETMAIN/FREEMAIN/STORAGE (GFS) Trace

GFS trace helps you identify problems related to requests to obtain or free storage. GFS trace uses the generalized trace facility (GTF) to collect data about storage requests, and places this data in USRF65 user trace records in the GTF trace. You can use IPCS to format the GTF trace records.

To turn GFS trace on or off, activate a DIAGxx parmlib member, either at IPL (specify DIAG=xx as a system parameter) or during normal processing (enter a SET DIAG=xx operator command). If you do not specify a DIAGxx parmlib member at IPL, the system processes the default member DIAG00, which turns GFS trace off. See *z/OS MVS System Commands* for more information about using SET DIAG=xx.

The DIAGxx parmlib member also allows you to specify:

- **GFS trace filtering data**, which limits GFS trace output according to:
 - Address space identifier (ASID)
 - Storage subpool
 - The length of the storage specified on a request
 - Storage key
- **GFS trace record data**, which determines the data to be placed in each record, according to one or more data items specified on the DATA keyword.

GFS trace degrades performance to a degree that depends on the current workload in the system. Therefore, it is a good idea to activate GFS trace only when you know there is a problem.

For information about how to request a GFS trace, see *z/OS MVS Diagnosis: Tools and Service Aids*.

Auxiliary Storage Overview

An installation needs auxiliary direct access storage devices (DASD) for housing all system data sets.

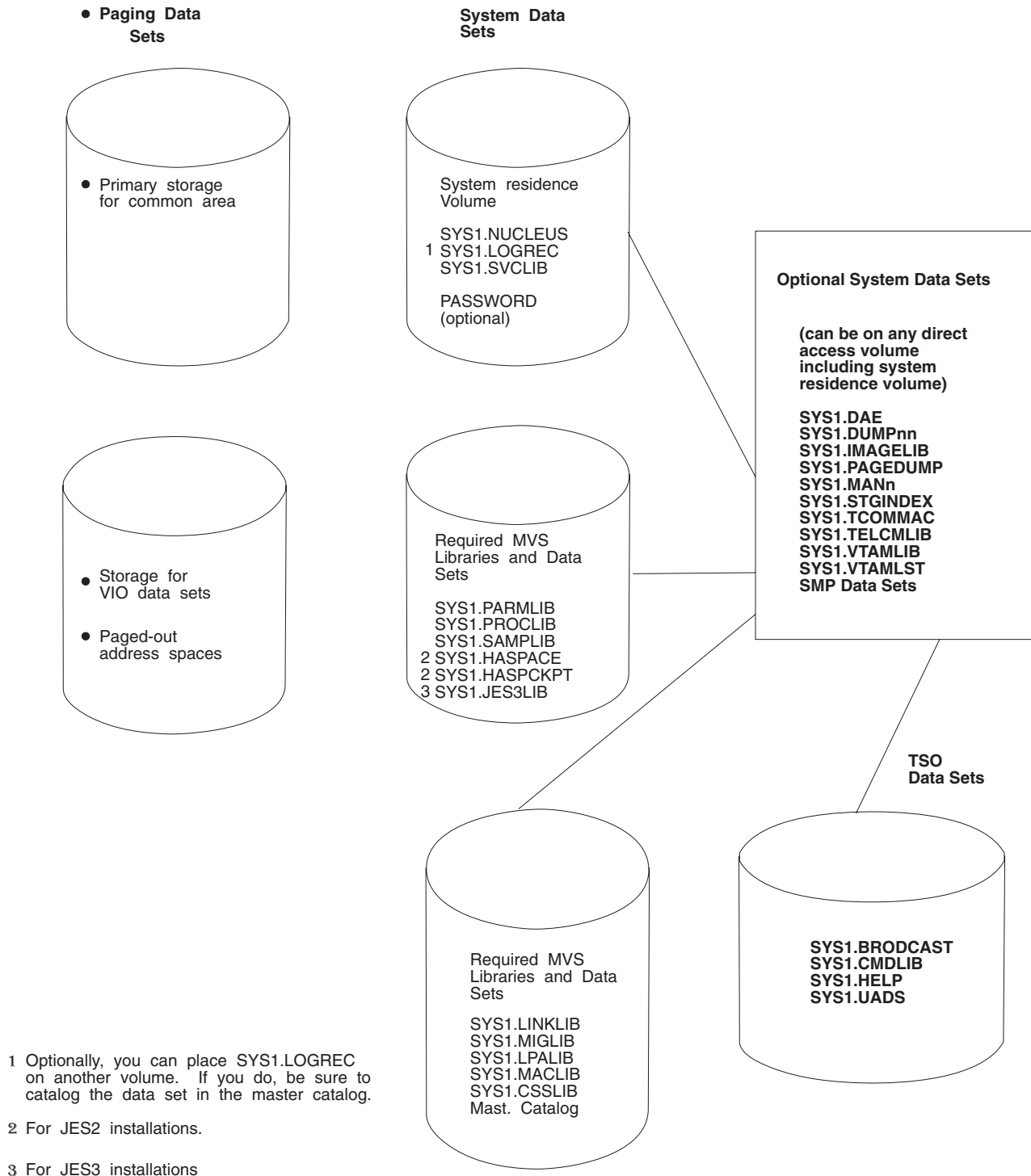
Enough auxiliary storage must be available for the programs and data that comprise the system. Auxiliary storage used to support basic system requirements has three logical areas (see Figure 1-3):

- System data set storage area.
- Paging data sets for backup of all pageable address spaces.

System Data Sets

Each installation must incorporate required system data sets into the system by allocating space for them on appropriate direct access devices during system installation. The DEFINE function of access method services is used to define both the storage requirements and the volume for each system data set. Some data sets

must be allocated on the system residence volume while some can be placed on other direct access volumes.



(Note that auxiliary storage need not be organized as shown above. This figure summarizes the makeup of auxiliary storage.)

Figure 1-3. Auxiliary Storage Requirement Overview

Paging Data Sets

Paging data sets contain the paged-out portions of all virtual storage address spaces. In addition, output to virtual I/O devices may be stored in the paging data sets. Before the first IPL, an installation must allocate sufficient space on paging data sets to back up the following virtual storage areas:

- Primary storage for the pageable portions of the common area
- Secondary storage for duplicate copies of the pageable common area
- The paged-out portions of all swapped-in address spaces - both system and installation
- Space for all address spaces that are, or were, swapped out
- VIO data sets that are backed by auxiliary storage.

Paging data sets are specified in IEASYSxx members of SYS1.PARMLIB. When this is done, any PAGE parameter in an IEASYSxx specified during IPL overrides any PAGE parameter in IEASYS00.

To add paging space to the system after system installation, the installation must use the access method services to define and format the new paging data sets. To add the new data sets to the existing paging space, either use the PAGEADD operator command or relPL the system, issuing the PAGE parameter at the operator's console. To delete paging space, use the PAGEDEL operator command.

During initialization, paging spaces are set up by merging the selected IEASYSxx parmlib member list of data set names with any names provided by the PAGE parameter (issued at the operator console) and any names from the previous IPL.

Directed VIO

When backed by auxiliary storage, VIO data set pages can be directed to a subset of the local paging data sets through *directed VIO*, which allows the installation to direct VIO activity away from selected local paging data sets and use these data sets only for non-VIO paging. With directed VIO, faster paging devices can be reserved for paging where good response time is important. The NONVIO system parameter, with the PAGE parameter in the IEASYSxx parmlib member, allows the installation to define those local paging data sets that are not to be used for VIO, leaving the rest available for VIO activity. However, if space is depleted on the paging data sets that were made available for VIO paging, the non-VIO paging data sets will be used for VIO paging.

The installation uses the DVIO keyword in the IEAOPTxx parmlib member to either activate or deactivate directed VIO. To activate directed VIO, the operator issues a SET OPT=xx command where the xx specifies the IEAOPTxx parmlib member that contains the DVIO=YES keyword; to deactivate directed VIO, xx specifies an IEAOPTxx parmlib member that contains the DVIO=NO keyword. The NONVIO parameter of IEASYSxx and the DVIO keyword of IEAOPTxx is explained more fully in *z/OS MVS Initialization and Tuning Reference*.

Primary and Secondary PLPA

During initialization, both primary and secondary PLPAs are established. The PLPA is established initially from the LPALST concatenation.

On the PLPA page dataset, ASM maintains records that have pointers to the PLPA and extended PLPA pages. During quick start and warm start IPLs, the system uses

the pointers to locate the PLPA and extended PLPA pages. The system then rebuilds the PLPA and extended PLPA page and segment tables, and uses them for the current IPL.

If CLPA is specified at the operator's console, indicating a cold start is to be performed, the PLPA storage is deleted and made available for system paging use. A new PLPA and extended PLPA is then loaded, and pointers to the PLPA and extended PLPA pages are recorded on the PLPA page dataset. CLPA also implies CVIO.

Virtual I/O Storage Space

Virtual I/O operations may send VIO dataset pages to the local paging dataset space. During a quick start, the installation uses the CVIO parameter to purge VIO dataset pages. During a warm start, the system can recover the VIO dataset pages from the paging space. If an installation wants to delete VIO page space reserved during the warm start, it issues the CVIO system parameter at the operator's console. CVIO applies only to the VIO dataset pages that are associated with journaled job classes. (The VIO journaling dataset contains entries for the VIO datasets associated with journaled job classes.) If there are no journaled job classes or no VIO journaling dataset, there is no recovery of VIO dataset pages. Instead, all VIO dataset pages are purged and the warm start is effectively a quick start.

If the SPACE parameter for a DD statement having a UNIT parameter, associated with a UNITNAME that was defined with having VIO=YES, is not specified, the default size is 10 primary and 50 secondary blocks with an average block length of 1000 bytes.

The cumulative number of page datasets must not exceed 256. This maximum number of 256 page datasets should follow these guidelines:

- There must be exactly one *PLPA* page dataset.
- There must be exactly one *common* page dataset.
- There must be at least one *local* page dataset, but no more than 253.

The actual number of pages required in paging datasets depends on the system load, including the size of the VIO datasets being created, the rate at which they are created, the rate at which they are deleted, and how much ESTORE is available and usable for VIO pages.

Improving Module Fetch Performance With LLA

You can improve the performance of module fetching on your system by allowing library lookaside (LLA) to manage your production load libraries. LLA reduces the amount of I/O needed to locate and fetch modules from DASD storage.

Specifically, LLA improves module fetch performance in the following ways:

- By maintaining (in the LLA address space) copies of the library directories the system uses to locate load modules. The system can quickly search the LLA copy of a directory in virtual storage instead of using costly I/O to search the directories on DASD.
- By placing (or **staging**) copies of selected modules in a virtual lookaside facility (VLF) data space (when you define the LLA class to VLF, and start VLF). The system can quickly fetch modules from virtual storage, rather than using slower I/O to fetch the modules from DASD.

LLA determines which modules, if staged, would provide the most benefit to module fetch performance. LLA evaluates modules as candidates for staging based on statistics it collects about the members of the libraries it manages (such as module size, frequency of fetches per module (fetch count), and the time required to fetch a particular module).

If necessary, you can directly influence LLA staging decisions through installation exit routines (CSVLLIX1 and CSVLLIX2). For information about coding these exit routines, see *z/OS MVS Installation Exits*.

LLA and Module Search Order

When a program requests a module, the system searches for the requested module in various system areas and libraries, in the following order:

1. Modules that were loaded under the current task (LLEs)
2. The job pack area (JPA)
3. Tasklib, steplib, joblib, or any libraries that were indicated by a DCB specified as an input parameter to the macro used to request the module (LINK, LINKX, LOAD, ATTACH, ATTACHX, XCTL or XCTLX).
4. Active link pack area (LPA), which contains the FLPA and MLPA
5. Pageable link pack area (PLPA)
6. SYS1.LINKLIB and libraries concatenated to it through the LNKLSTxx member of parmlib. ("Placing Modules in the System's Search Order for Programs" on page 1-14 explains the performance improvements that can be achieved by moving modules from the LNKLST concatenation to LPA.)

When searching TASKLIBs, STEPLIBs, JOBLIBs, a specified DCB, or the LNKLST concatenation for a module, the system searches each data set directory for the first directory entry that matches the name of the module. The directory is located on DASD with the data set, and is updated whenever the module is changed. The directory entry contains information about the module and where it is located in storage. (For more information, see the "Program Management" topic in the *z/OS MVS Programming: Assembler Services Guide*.)

As mentioned previously, LLA caches, in its address space, a copy of the directory entries for the libraries it manages. For modules that reside in LLA-managed libraries, the system can quickly search the directories in virtual storage instead of using I/O to search the directories on DASD.

Planning To Use LLA

To use LLA, do the following:

1. Determine which libraries LLA is to manage
2. Code members of parmlib (see "Coding the Required Members of Parmlib" on page 1-35)
3. Control LLA through operator commands (see "Controlling LLA and VLF Through Operator Commands" on page 1-37).

When determining which data sets LLA is to manage, try to limit these choices to production load libraries that are rarely changed. Because LLA manages LNKLST libraries by default, you need only determine which non-LNKLST libraries LLA is to manage. If, for some reason, you do not want LLA to manage particular LNKLST libraries, you must explicitly remove such libraries from LLA management (as described in "Removing Libraries From LLA Management" on page 1-39).

Using VLF With LLA

Because you obtain the most benefit from LLA when you have both LLA and VLF functioning, you should plan to use both. When used with VLF, LLA reduces the I/O required to fetch modules from DASD by causing selected modules to be staged in VLF data spaces. LLA does not use VLF to manage library directories. When used without VLF, LLA eliminates only the I/O the system would use in searching library directories on DASD.

LLA Notes

1. All LLA-managed libraries must be cataloged. This includes LNKST libraries. A library must remain cataloged for the entire time it is managed by LLA. Please see “Recataloging LLA-Managed Data Sets While LLA is Active” on page 1-42 for additional information about recataloging LLA-managed libraries.
2. The benefits of LLA load module caching applies only to modules that are retrieved through the following macros: LINK, LINKX, LOAD, ATTACH, ATTACHX, XCTL and XCTLX.
3. LLA does not stage load modules in overlay format. LLA manages the directory entries of overlay format modules, but the modules themselves are provided through program fetch. If you want to make overlay format modules eligible for staging, you must re-linkedit the modules as non-overlay format. These reformatted modules might occupy more storage when they execute and, if LLA does not stage them, might take longer to be fetched.

Coding the Required Members of Parmlib

LLA and VLF are associated with parmliib members, as follows:

- Use the CSVLLAxx member to identify the libraries that LLA is to manage.
- Use the COFVLFxx member to extend VLF services to LLA.

This section provides guidance information for coding the keywords of CSVLLAxx. For information about the required syntax and rules for coding CSVLLAxx and COFVLFxx, see *z/OS MVS Initialization and Tuning Reference*.

Coding CSVLLAxx

Use CSVLLAxx to specify which libraries LLA is to manage and how it is to manage them.

IBM does not provide a default CSVLLAxx member (such as CSVLLA00). If you do not supply a CSVLLAxx member, LLA will, by default, manage only the libraries that are accessed through the LNKST concatenation. If you supply a CSVLLAxx member, LLA manages the libraries you specify in CSVLLAxx (on the LIBRARIES keyword) in addition to any libraries that are accessed through the LNKST concatenation. Because LLA manages the LNKST libraries by default, it is not necessary to specify any LNKST libraries on the LIBRARIES keyword.

Using Multiple CSVLLAxx Members: To use more than one CSVLLAxx member at a time, specify the additional members to be used on the PARMLIB and SUFFIX keywords in the original CSVLLAxx member. The CSVLLAxx members pointed to by the PARMLIB and SUFFIX keywords must not point back to the original member, nor to each other.

You can use the PARMLIB and SUFFIX keywords to specify CSVLLAxx members that reside in data sets other than PARMLIB. You can then control LLA's specifications without having update access to PARMLIB.

You can also use the PARMSUFFIX parameter of the CSVLLAxx parmlib member to specify additional CSVLLAxx members. PARMSUFFIX differs from the PARMLIB(dsn) SUFFIX(xx) statement in that no data set name is specified. PARMSUFFIX searches the logical parmlib for the CSVLLAxx member.

Example of Multiple CSVLLAxx Members: Let's say you want two PARMLIBS (IMS.PARMLLA and AMVS.LLAPARMS) so that a TSO REXX™ exec can automatically activate a new module in LNKST when it has copied the new module into a LNKST library.

Do the following:

```
START LLA,LLA=IM,SUB=MSTR with CSVLLAIM as:
```

```
FREEZE(-LNKST-)
PARMLIB(IMS.PARMLLA)
SUFFIX(IA)
PARMLIB(AMVS.LLAPARMS)
SUFFIX(RX)
```

where:

AMVS.LLAPARMS (CSVLLARX) would contain the latest update requested by the REXX exec, such as:

```
REMOVE(...)/LIBRARIES(...)MEMBERS...
or
PARMSUFFIX(...)
or
PARMLIB(...) SUFFIX(...)
```

The REXX exec and either:

- use multiple members and use the PARMSUFFIX to identify them, or
- move the old CSVLLARX to CSVLLARn before building the new one.

Storing Program Objects in PDSEs: In VMS/ESA 4.3 with DFSMS 1.1.0 and SMS active, you can produce a program object, and executable program unit that can be stored in a partitioned data set extended (PDSE) program library. Program objects resemble load modules in function, but have fewer restrictions and are stored in PDSE libraries instead of PDS libraries. PDSE libraries that are to be managed by LLA must contain program objects. LLA manages both load and program libraries.

Coding COFVLFxx

To have VLF stage load modules from LLA-managed libraries, you can use the default COFVLFxx member that is shipped in parmlib (COFVLF00), or, optionally, an installation-supplied COFVLFxx member. The installation-supplied member must contain a CLASS statement for LLA (see COFVLF00 for an example).

If you modify the COFVLFxx parmlib member, you must stop and restart VLF to make the changes effective.

If you want to use an installation-supplied COFVLFxx member instead of COFVLF00, do the following:

- Specify a CLASS statement for LLA in the alternative COFVLFxx member
- Specify the suffix of the alternative COFVLFxx member on the START VLF command. Otherwise, the system uses COFVLF00 by default.

For information about the required syntax and rules for coding the COFVLFxx member, see *z/OS MVS Initialization and Tuning Reference*.

Controlling LLA and VLF Through Operator Commands

Use the following commands to control LLA:

- START LLA and START VLF
- STOP LLA and STOP VLF
- MODIFY LLA.

This section explains how to use commands to control LLA. For information about the required syntax and rules for entering commands, see *z/OS MVS System Commands*.

Starting LLA

The START LLA command is included in the IBM-supplied IEACMD00 member of parmlib. Therefore, the system automatically starts LLA when it reads the IEACMD00 member during system initialization.

Although the system automatically starts LLA, it does not, by default, activate a CSVLLAxx member. To activate a CSVLLAxx member at system initialization, specify the suffix (xx) of the CSVLLAxx member in either of the following places:

- In the LLA procedure in SYS1.PROCLIB, as shown:

```
//LLA PROC LLA=xx  
//LLA EXEC PGM=CSVLLCRE,PARM='LLA=&LLA'
```

where 'xx' matches the suffix on the CSVLLAxx member to be used.

- On the START LLA command in the IEACMD00 member, as shown:

```
COM='START LLA,SUB=MSTR,LLA=xx'
```

where 'xx' matches the suffix on the CSVLLAxx member to be used.

You should not set a region limit LLA, either by setting a region size or by an IEFUSI exit. This will avoid storage exhaustion abends in the LLA address space.

If you do not supply a CSVLLAxx member, LLA will by default manage only the libraries that are accessed through the LNKST concatenation.

When started, LLA manages both the libraries specified in the CSVLLAxx member as well as the libraries in the LNKST concatenation.

If you wish to use a parmlib data set other than SYS1.PARMLIB for LLA, specify the data set LLA is to use on an //IEFPARM DD statement in the LLA procedure in PROCLIB.

Starting LLA After System Initialization: IBM recommends that you specify SUB=MSTR on the START LLA command to prevent LLA from failing if JES fails. For example:

```
START LLA,SUB=MSTR,LLA=xx
```

where 'xx' matches the suffix on the CSVLLAxx member to be used, if any.

Starting VLF

LLA provides better performance when VLF services are available, so it is better (although not necessary) to start VLF before LLA. However, the operation of LLA does not depend on VLF.

To allow VLF to be started through the START command, create a VLF procedure, or use the following procedure, which resides in SYS1.PROCLIB:

```
//VLF      PROC   NN=00  
//VLF      EXEC   PGM=COFMINIT,PARM='NN=&NN'
```

When you issue the START VLF command, the VLF procedure activates the IBM-supplied COFVLF00 member, which contains a CLASS statement for LLA.

Stopping LLA and VLF

You can stop LLA and VLF through STOP commands. Note that stopping VLF purges all the data in VLF data spaces for all classes defined to VLF (including LLA), and will slow performance.

If LLA or VLF is stopped (either by a STOP command or because of a system failure), you can use a START command to reactivate the function.

Modifying LLA

You can use the MODIFY LLA command to change LLA dynamically, in either of the following ways:

- **MODIFY LLA,REFRESH.** Rebuilds LLA's directory for the entire set of libraries managed by LLA. This action is often called a *complete refresh* of LLA.
- **MODIFY LLA,UPDATE=xx.** Rebuilds LLA's directory only for specified libraries or modules. xx identifies the CSVLLAxx member that contains the names of the libraries for which directory information is to be refreshed. This action is often called a *selective refresh* of LLA. (For details, see "Identifying Members for Selective Refreshes.")

When an LLA-managed library is updated, the version of a module that is located by a directory entry saved in LLA will differ from the version located by the current directory entry on DASD for that module. If you update a load module in a library that LLA manages, it is a good idea to follow the update by issuing the appropriate form of the MODIFY LLA command to refresh LLA's cache with the latest version of the directory information from DASD. Otherwise, the system will continue to use an older version of a load module.

Note: Applications can use the LLACOPY macro to refresh LLA's directory information. For information about the LLACOPY macro, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Identifying Members for Selective Refreshes: In CSVLLAxx, specify the MEMBERS keyword to identify members of LLA-managed libraries for which LLA-cached directory entries are to be refreshed during selective refreshes of LLA. If you issue the MODIFY LLA,UPDATE=xx command to select a CSVLLAxx member that has libraries specified on the MEMBERS keyword, LLA will update its directory for each of the members listed on the MEMBERS keyword.

Selectively refreshing LLA directory allows updated LLA-managed modules to be activated without activating other changed LLA-managed modules. Selective LLA refresh also avoids the purging and restaging of modules that have not changed. When a staged module is refreshed in the LLA directory, LLA purges the copy of the module in the virtual lookaside facility (VLF) data space and may then restage the module into VLF.

For more information about specifying the MEMBERS keyword, see the description of the CSVLLAxx member in *z/OS MVS Initialization and Tuning Reference*.

Removing Libraries From LLA Management

In CSVLLAxx, specify the REMOVE keyword for libraries that are to be removed dynamically from LLA management. If you issue the MODIFY LLA,UPDATE=xx command (selective refresh) to select a CSVLLAxx member that lists libraries on the REMOVE keyword, LLA no longer provides directory entries or staged modules for these libraries, regardless of whether the libraries are included in the LNKST concatenation. (Note that you cannot use REMOVE to change the order or contents of the LNKST concatenation itself.)

You can also use the MODIFY LLA,REFRESH command (complete refresh) to remove libraries from LLA management. This command rebuilds the entire LLA directory, rather than specified entries in LLA's directory. To limit the adverse effects on performance caused by an LLA refresh, whenever possible, use a selective refresh of LLA instead of a complete refresh, or stopping and restarting LLA.

In any case, when LLA directory entries are refreshed, LLA discards directory information of the associated module and causes VLF (if active) to remove the module from the VLF cache. This reduces LLA's performance benefit until LLA stages them again. Because LLA stages modules using the cached directory entries, you should refresh LLA whenever a change is made to an LLA-managed data set.

The MODIFY LLA command does not reload (or refresh) modules that are already loaded into virtual storage, such as modules in long-running or never-ending tasks.

For more information about specifying the REMOVE keyword, see the description of the CSVLLAxx member in *z/OS MVS Initialization and Tuning Reference*.

Modifying Shared Data Sets

You can allow two or more systems to share access to the same library directory. When modifying or stopping LLA in this case, your changes must take effect simultaneously on all systems that share access to the directory.

In cases where you simply want to update an LLA-managed data set, it is easier to remove the data set from LLA management and update it, rather than stopping LLA on all systems. To do so, enter a MODIFY LLA,UPDATE=xx command on each system that shares access to the data set, where 'xx' identifies a CSVLLAxx member that specifies, on the REMOVE keyword, the data set to be removed from LLA management.

When you have completed updating the data set, enter the MODIFY LLA,UPDATE=xx command again, this time specifying a CSVLLAxx parmlib member in which the keyword LIBRARIES specifies the name of the data set.

In any case, whenever multiple systems share access to an LLA-managed data set, STOP, START, or MODIFY commands must be entered for LLA on all the systems.

Using the FREEZEINOFREEZE Option

For an LLA-managed library, use the FREEZEINOFREEZE option to indicate whether the system is to search the LLA-cached or DASD copy of a library directory. With FREEZEINOFREEZE, which you code in the CSVLLAxx member, you specify on a library-by-library basis which directory copy the system is to search, as follows:

- If you specify FREEZE, the system uses the copy of the directory that is maintained in the LLA address space (the library is "frozen").

- If you specify **NOFREEZE**, the system searches the directory that resides in DASD storage.

The system always treats libraries in the LNKST concatenation as frozen. Therefore, you need only specify the **FREEZE** option for non-LNKST libraries, or for LNKST libraries that are referenced through TASKLIBs, STEPLIBs, or JOBLIBs.

When an LLA-managed library is frozen, the following is true:

- Users of the library always receive versions of the library's modules that are pointed to by the LLA-cached directory.
- Users do not see any updates to the library until LLA is refreshed. If a user does multiple linkedits to a member in a **FREEZE** data set, the base for each subsequent linkedit does not include the previous linkedits; the base is the LLA version of the member.
- If the version of a requested module matches the version of the module in VLF, the users receive the module from VLF. Otherwise, users receive the module from DASD.

To take full advantage of LLA's elimination of I/O for directory search, specify **FREEZE** for as many read-only or infrequently updated libraries as appropriate.

Having **NOFREEZE** in effect for an LLA-managed library means that your installation does not eliminate I/O while searching the library's directory. However, LLA can still improve performance when the system fetches load modules from the VLF data space instead of DASD storage.

Table 1-2 summarizes the affects of the **FREEZE/NOFREEZE** option on directory search and module fetch.

Table 1-2. FREEZE/NOFREEZE Processing

Action	FREEZE or NOFREEZE in Effect	LNKST Libraries Accessed From LNKST	LNKST Libraries Accessed Outside LNKST	Other Non-LNKST Libraries
Directory Search	FREEZE	LLA directory is used.	LLA directory is used.	LLA directory is used.
	NOFREEZE	LLA directory is used.	DASD directory is used (I/O occurs)	DASD directory is used (I/O occurs)
Module Fetch	FREEZE	Requestor receives LLA version of module (from VLF data space or DASD).	Requestor receives LLA version of module (from VLF data space or DASD).	Requestor receives LLA version of module (from VLF data space or DASD).
	NOFREEZE	Requestor receives LLA version of module (from VLF data space or DASD).	Requestor receives most current version of module (from DASD or VLF data space, if staged).	Requestor receives most current version of module (from DASD or VLF data space, if staged).

You can change the FREEZEINOFREEZE status of an LLA-managed library at any time through the MODIFY LLA command. Changing a library from NOFREEZE to FREEZE also causes a refresh of the directory information for that library (note that when a library is refreshed, all of its modules are destaged from the VLF data space, which will slow performance until new versions are staged).

For more information about specifying the FREEZEINOFREEZE option, see the description of the CSVLLAxx member in *z/OS MVS Initialization and Tuning Reference*.

Changing LLA-Managed Libraries

After changing a module in an LLA-managed library, IBM recommends that you refresh LLA for the library. A module that is changed and has a new location is not considered for staging until that member is refreshed.

The recommended way to make updates in the production system is to use IEBCOPY under FREEZE mode. The member to be updated should be copied to another data set, the linkedits run against the second data set and then the updated member can be copied back to the LLA-managed data set. If LLA-managed production libraries must be updated directly, LLA should be refreshed to manage the data set in NOFREEZE mode.

LLA ENQ Consideration: By default, LLA allocates the libraries it manages as DISP=SHR. This means that if a job attempts to allocate an LLA-managed library as DISP=OLD, the job is enqueued until LLA is stopped or the library is removed from LLA management. Before adding a library to the libraries that LLA manages, review and, if necessary, change the jobs that reference the library.

Using the GET_LIB_ENQ Keyword: The GET_LIB_ENQ keyword in the CSVLLAxx member allows you to prevent LLA from obtaining an exclusive enqueue on the libraries it manages. If you specify GET_LIB_ENQ (NO), your installation's jobs can update, move, or delete LLA-managed libraries while other users are accessing the libraries. GET_LIB_ENQ (NO) is generally not recommended, however, because of the risks it poses to data set integrity.

Compressing LLA-Managed Libraries: If you compress an LLA-managed library, LLA continues to provide the obsolete directory entries. For members that have been staged to the VLF data space, the system will operate successfully. If the member is not currently staged, however, the cached obsolete directory entry can be used to fetch the member at the old TTR location from DASD.

Because using obsolete directory entries can cause such problems as abends, breaches of system integrity, and system failures, use the following procedure to compress LLA-managed libraries:

1. Issue a MODIFY LLA,UPDATE=xx command, where the CSVLLAxx parmlib member includes a REMOVE statement identifying the library that needs to be compressed.
2. Compress the library
3. Issue a MODIFY LLA,UPDATE=xx command, where the CSVLLAxx parmlib member includes a LIBRARIES statement to return the compressed library to LLA management.

This procedure causes all members of that library to be discarded from the VLF data space. The members are then eligible to be staged again.

Recataloging LLA-Managed Data Sets While LLA is Active

LLA dynamically allocates the library data sets it manages. During unallocation (when LLA is stopped or the system is reIPLed), the system recatalogs the library data sets to the volume that was current at LLA initialization. Therefore, to recatalog an LLA-managed library data set while LLA is active, do the following:

1. Remove the library data set from LLA. (Issue a MODIFY LLA,UPDATE=xx command, where xx identifies the suffix of the CSVLLAxx parmlib member that includes a REMOVE statement that identifies the library data set to be removed from LLA management.)
2. Recatalog the library data set.
3. Return the library data set to LLA. (Issue a MODIFY LLA,UPDATE=xx command, where xx identifies the suffix of the CSVLLAxx parmlib member that includes a LIBRARIES statement that identifies the recataloged library data set to be returned to LLA management.) Recataloged LNKST libraries cannot be put back into LLA management. This causes fetch failures.

Allocation Considerations

Before a job can execute, the operating system must set aside the devices, and space on the devices, for the data that is to be read, merged, sorted, stored, punched, or printed. In MVS, the “setting-aside” process is called **allocation**.

MVS assigns **units** (devices), **volumes** (space for data sets), and **data sets** (space for collections of data) according to the *data definition* (DD) and *data control block* (DCB) information included in the JCL for the job step.

When the data definition or DCB information is in the form of text units, the allocation of resources is said to be *dynamic*. Dynamic allocation means you are requesting the system to allocate and/or deallocate resources for a job step while it is executing. For details on the use of dynamic allocation, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Serialization of Resources During Allocation

When the system is setting aside non-sharable devices, volumes and data sets for a job or a step, it must prevent any other job from using those resources during the allocation process. To prevent a resource from changing status while it is being allocated to a job, the system uses **serialization**. Serialization during allocation causes jobs to wait for the resources and can have a major impact on system performance. Therefore, the system attempts to minimize the amount of time lost to serialization by providing a specific order of allocation processing. See Table 1-3.

Knowing the order in which the system chooses devices, you can improve system performance by making sure your installation’s jobs request resources that require the least possible serialization.

The system processes resource allocation requests in the order shown in Table 1-3. As you move down the list, the degree of serialization – and processing time – increases.

Table 1-3. Processing Order for Allocation Requests Requiring Serialization

KINDS OF ALLOCATION REQUESTS	SERIALIZATION REQUIRED
Requests requiring no specific units or volumes; for example, DUMMY, VIO, and subsystem data sets.	No serialization.

Table 1-3. Processing Order for Allocation Requests Requiring Serialization (continued)

KINDS OF ALLOCATION REQUESTS	SERIALIZATION REQUIRED
Requests for sharable units: DASD that have permanently resident or reserved volumes mounted on them.	No serialization.
Teleprocessing devices.	Serialized on the requested devices.
Pre-mounted volumes, and devices that do not need volumes.	Serialized on the group(s) of devices eligible to satisfy the request. A single generic device type is serialized at a time.
Online, nonallocated devices that need the operator to mount volumes.	Serialized on the group(s) of devices eligible to satisfy the request. A single generic device type is serialized at a time.
All other requests: offline devices, nonsharable devices already allocated to other jobs.	Serialized on one or more groups of devices eligible to satisfy the request. A single generic device type is serialized at a time.

Improving Allocation Performance

You can contribute to the efficiency of allocation processing throughout your installation in several ways:

- For **devices**:
 - Use the **device preference table**, specified through the Hardware Configuration Definition. (HCD) to set up the order for device allocation. See the *z/OS MVS Device Validation Support* appendix for a listing of the device preference table. installation's devices as *esoteric groups*, and to group them for selection by allocation processing.
- Use the **eligible device table** (EDT) to identify your
For more information on the device preference table and the EDT, see *z/OS HCD Planning*.
- For **volumes**, use the **VATLSTxx** members of SYS1.PARMLIB to specify volume attributes at IPL.
- For **data sets**, you can specify the JCL used for your installation's applications according to the device selection criteria you have set up through the HCD process and IPL.

The Volume Attribute List

In MVS, each online device in the installation has a **mount attribute** and each mounted volume has a **use attribute**.

The **mount** attributes determine when the volume on that device should be removed. This information is needed when selecting a device and during step unallocation processing. The **mount** attributes are:

- Permanently resident
- Reserved
- Removable.

The **use** attributes determine the type of nonspecific requests eligible to that volume. The **use** attributes are:

- Public
- Private
- Storage.

Allocation routines use the volumes' **mount** and **use** attributes in selecting devices to satisfy allocation requests. Thoughtful selection of the use and mount attributes is

important to the efficiency of your installation. For example, during allocation, data sets on volumes marked permanently resident or reserved are selected first because they require no serialization, thus minimizing processing time.

During system initialization, you can assign volume attributes to direct access volumes by means of the **volume attribute list**. The volume attribute list is defined at IPL time using the VATLSTxx member of SYS1.PARMLIB.

After an IPL, you can assign volume attributes to both direct access and tape volumes using the MOUNT command. The USE= parameter on the MOUNT command defines the use attribute the volume is to have; a mount attribute of *reserved* is automatic. For the details of using the MOUNT command, see *z/OS MVS System Commands*.

When volumes are not specifically assigned a use attribute in the VATLSTxx member or in a MOUNT command, the system assigns a default. You can specify this default using the VATDEF statement in VATLSTxx. If you do not specify VATDEF, the system assigns a default of *public*. For details on including a volume attribute list in IEASYSxx, and on coding the VATLSTxx parmlib member itself, see *z/OS MVS Initialization and Tuning Reference*.

Use and Mount Attributes

Every volume is assigned use and mount attributes through an entry in the VATLSTxx member at IPL, a MOUNT command, or by the system in response to a DD statement.

The relationships between use and mount attributes are complex, but logical. The kinds of devices available in an installation, the kinds of data sets that will reside on a volume, and the kinds of uses the data sets will be put to, all have a bearing on the attributes assigned to a volume. Generally, the operating system establishes and treats volume attributes as outlined below.

Use attributes

- **Private** — meaning the volume can only be allocated when its volume serial number is explicitly or implicitly specified.
- **Public** — meaning the volume is eligible for allocation to a temporary data set, provided the request is not for a specific volume and PRIVATE has not been specified on the VOLUME parameter of the DD statement.

Both tape and direct access volumes are given the public use attribute.

A public volume may also be allocated when its volume serial number is specified on the request.

- **Storage** — meaning the volume is eligible for allocation to both temporary and non-temporary data sets, when no specific volume is requested and PRIVATE is not specified. Storage volumes usually contain non-temporary data sets, but temporary data sets that cannot be assigned to public volumes are also assigned to storage volumes.

Mount attributes

- **Permanently resident** — meaning the volume cannot be demounted. Only direct access volumes can be permanently resident. The following volumes are always permanently resident:
 - All volumes that cannot be physically demounted, such as drum storage and fixed disk volumes
 - The IPL volume

- The volume containing the system data sets

In the VATLSTxx member, you can assign a permanently-resident volume any of the three use attributes. If you do not assign a use attribute to a permanently-resident volume, the default is public.

- **Reserved** — meaning the volume is to remain mounted until the operator issues an UNLOAD command.

Both direct access and tape volumes can be reserved because of the MOUNT command; only DASD volumes can be reserved through the VATLSTxx member.

The reserved attribute is usually assigned to a volume that will be used by many jobs to avoid repeated mounting and demounting.

You can assign a reserved *direct access* volume any of the three use attributes, through the USE parameter of the MOUNT command or the VATLSTxx member, whichever is used to reserve the volume.

A reserved *tape* volume can only be assigned the use attributes of private or public.

- **Removable** — meaning that the volume is not permanently resident or reserved. A removable volume can be demounted either after the end of the job in which it is last used, or when the unit it is mounted on is needed for another volume.

You can assign the use attributes of private or public to a removable *direct access* volume, depending on whether VOLUME=PRIVATE is coded on the DD statement: if this subparameter is coded, the use attribute is private; if not, it is public.

You can assign the use attributes of private or public to a removable *tape* volume under one of the following conditions:

- **Private**

- The PRIVATE subparameter is coded on the DD statement.
- The request is for a specific volume.
- The data set is nontemporary (not a system-generated data set name, and a disposition other than DELETE).

Note: The request must be for a **tape only** data set. If, for example, an esoteric group name includes both tape and direct access devices, a volume allocated to it will be assigned a use attribute of public.

- **Public**

- The PRIVATE subparameter is not coded on the DD statement.
- A nonspecific volume request is being made.
- The data set is temporary (a system-generated data set name, or a disposition of DELETE).

Table 1-4 summarizes the mount and use attributes and how they are related to allocation requests.

Table 1-4. Summary of Mount and Use Attribute Combinations

Volume State	Temporary Data Set	Nontemporary data set	How Assigned	How Demounted
	Type of Volume Request			
Public and Permanently Resident ¹	Nonspecific or Specific	Specific	VATLSTxx entry or by default	Always mounted

Table 1-4. Summary of Mount and Use Attribute Combinations (continued)

Volume State	Temporary Data Set	Nontemporary data set	How Assigned	How Demounted
	Type of Volume Request			
Private and Permanently Resident ¹	Specific	Specific	VATLSTxx entry	Always mounted
Storage and Permanently Resident ¹	Nonspecific or Specific	Nonspecific or Specific	VATLSTxx entry	Always mounted
Public and Reserved (Tape and direct access)	NonSpecific or Specific	Specific	Direct access: VATLSTxx entry or MOUNT command Tape: MOUNT command	UNLOAD or VARY OFFLINE commands
Private and Reserved (Tape and direct access)	Specific	Specific	Direct access: VATLSTxx entry or MOUNT command Tape: MOUNT command	UNLOAD or VARY OFFLINE commands
Storage and Reserved ¹	Nonspecific or Specific	Nonspecific or Specific	VATLSTxx entry or MOUNT command	UNLOAD or VARY OFFLINE commands
Public and Removable (Tape and direct access)	Nonspecific or Specific	Specific	VOLUME=PRIVATE is not coded on the DD statement. (For tape, nonspecific volume request and a temporary data set also cause this assignment.)	When unit is required by another volume; or by UNLOAD or VARY OFFLINE commands.
Private and Removable (Tape and direct access)	Specific	Specific	VOLUME=PRIVATE is coded on the DD statement. (For tape, a specific volume request or a nontemporary data set also cause this assignment).	At job termination for direct access; at step termination or dynamic unallocation for tape (unless VOL=RETAIN or a disposition of PASS was specified); or when the unit is required by another volume.

The Nonsharable Attribute

Some allocation requests imply the exclusive use of a direct access device while the volume is demounted and/or mounted. The system assigns the **non-sharable attribute** to volumes that might require demounting during step execution.

When a volume is thus made non-sharable, it cannot be assigned to any other data set until the non-sharable attribute is removed at the end of step execution.

The following types of requests cause the system to automatically assign the non-sharable attribute to a volume:

- A specific volume request that specifies more volumes than devices.
- A nonspecific request for a *private* volume that specifies more volumes than devices.
- A volume request that includes a request for unit affinity to a preceding DD statement, but does not specify the same volume for the data set. For more information, see the discussion of unit affinity in *z/OS MVS JCL Reference*.
- A request for deferred mounting of the volume on which a requested data set resides.

Except for one situation, the system will NOT assign the non-sharable attribute to a permanently-resident or reserved volume. The exception occurs when the allocation request is for more volumes than units, and one of the volumes is reserved. The reserved volume is to share a unit with one or more *removable* volumes, which precede it in the list of volume serial numbers.

For example:

```
DSN=BCA.ABC,VOL=SER=(A,B),UNIT=DISK
```

where volume A is removable and
volume B is reserved

In this example, *both* volumes are assigned the non-sharable attribute; neither of them can be used in another job at the same time.

To avoid this situation, do one of the following:

- Specify the same number of volumes as units
- Specify parallel mounting
- Set the mount attribute of volume A as resident or reserved.

System Action: Table 1-5 shows the system action for sharable and non-sharable requests.

Table 1-5. Sharable and Nonsharable Volume Requests

The Request is:	The Volume is Allocated:	
	Sharable	Nonsharable
Sharable	allocate the volume	wait ¹
Nonsharable	wait ¹	wait ¹
¹ The operator has the option of failing the request. The request will always fail if waiting is not allowed.		

For more detailed information on how an application's JCL influences the processing of allocation requests, see *z/OS MVS JCL Reference*.

For details on how dynamic allocation affects the use attributes of the volumes in your installation, see *z/OS MVS Programming: Assembler Services Guide*.

Chapter 2. Auxiliary Storage Management Initialization

This chapter gives guidelines for the effective use of page data sets. Additional information on this subject appears under PAGE, NONVIO, and PAGTOTL parameters of parmlib member IEASYSxx in *z/OS MVS Initialization and Tuning Reference*.

Page Operations

The paging controllers of the auxiliary storage manager (ASM) attempt to maximize I/O efficiency by incorporating a set of algorithms to distribute the I/O load as evenly as is practical. In addition, every effort is made to keep the system operable in situations where a shortage of a specific type of page space exists. The following topics discuss some of the algorithms ASM uses to achieve these ends.

Paging Operations and Algorithms

To page efficiently and expediently, ASM divides the pages of the system into classes, namely PLPA, common, and local. Contention is reduced when these classes of pages are placed on different physical devices. Multiple local page data sets are recommended. Although the system requires only one local page data set, performance improvement can be obtained when local page data sets are distributed across more than one device, even though some devices may be large enough to hold the entire amount of necessary page space. The PLPA and common page data sets are both required data sets, and there can be only one of each. Spillage back and forth between the PLPA and common page data sets is permissible, but, in the interest of performance, only spilling from PLPA to common should be permitted.

The general intent of the ASM algorithms for page data set selection and channel program construction is to:

- *Use all available local page data sets:* When ASM writes a group of data, it selects a local page data set in a circular order within each type of device, considering the availability of free space and device response time.
When ASM selects a data set, the paging data sets that reside on Parallel Access Volume (PAV) devices are examined first because of reliability and performance characteristics. Because preference is given to PAV devices, it is normal to have a higher usage of PAV data sets as compared to non-PAV data sets.
- *Write group requests to contiguous slots:* ASM selects contiguous space in local page data sets on moveable-head devices to receive group write requests. For certain types of requests, ASM's slot allocation algorithm tries to select sequential (contiguous) slots within a cylinder. The reason for doing this is to shorten the I/O access time needed to read or write a group of requests. For other types of requests (such as an individual write request), or if there are no sequential slots, ASM selects any available slots.
- *Limit the apparent size of local page data sets to reduce seek time:* If possible, ASM concentrates group requests and individual requests within a subset of the space allocated to a local page data set. In using the suspend/resume function, ASM can monopolize a direct access storage device for its I/O and improve its performance for any of the requests going to page data sets.

Page Data Set Sizes

Page data set sizes can affect system performance. Note the following recommendations:

- *PLPA data set.* The total combined size of the PLPA page data set and common page data set can not exceed the size of the PLPA (and extended PLPA) plus the amount specified for the CSA and the size of the MLPA. In defining the size of these data sets, a reasonable starting value might be four megabytes for PLPA and 20 megabytes for common, as spilling will occur if the PLPA data set becomes full.
RMF reports can be used to determine the size requirements for these data sets. During system initialization, check the size of the page data sets in the page data set activity report. If the slots used values are very close to the slots allocated value, the size should be enlarged. For more information, see the report of the page data set activity in *z/OS RMF Report Analysis*.
- *Common page data set.* The common page data set should be large enough to contain all of the common area pages plus room for any expected PLPA spill. Although it is possible for the common page data set to spill to the PLPA page data set, this situation should not be allowed to occur because it may heavily impact performance. As noted for the PLPA data set, a reasonable starting size for the common page data set might be twenty megabytes.
- *Local page data sets.* The local page data sets must be large enough to hold the private area and VIO pages that can not be contained in real storage.

Page Data Set Protection

The page data set protection feature was introduced in z/OS V1R3 to help guard you from unintentionally IPLing with a page data set that is already in use. It does this by formatting and maintaining a status information record at the beginning of each page data set and by using an ENQ to serialize usage of the data sets.

The page data set protection feature prevents two systems from accidentally using the same physical data set. However, it is not possible to prevent the same data sets from being used when:

- the request to use the data set comes from a system outside of the GRS Ring/Star configuration.
- the installation has excluded the data sets from multi-system serialization. See *z/OS MVS Planning: Global Resource Serialization* for more information on SYSTEMS Exclusion RNL.

Page data sets are protected by a two-tier mechanism using:

- “SYSTEMS level ENQ”.
- “Status Information Record” on page 2-3.

SYSTEMS level ENQ

Page data sets are protected with a SYSTEMS level ENQ that contains the name of the page data set and the volser it resides on. The qname used on the ENQ is SYSZILRD, and the rname used on the ENQ is the dsname + volser. This ENQ is obtained during master scheduler initialization for the IPL-specified page data sets. For example, the ENQ would be obtained from the definitions in the IEASYSxx parmlib member. The ENQ is also obtained whenever a page data set is added or replaced after IPL.

A warning-level message is issued if the ENQ cannot be obtained successfully during IPL for the IPL-specified page data sets. Processing for the command is terminated if the ENQ cannot be obtained for a page data set that is being added or replaced with the PAGEADD or PAGEDEL command.

Status Information Record

A data set status information record is written to every page data set. The status record identifies the system using the data set.

The status record is validated during IPL. If the record indicates that the data set is in use by another system, message ILR030A is issued and the system waits for an operator response to allow or disallow use of the data set.

Note that this feature does not offer full protection in the case of a page data set that was defined on, or was last used by a pre-z/OS V1R3 environment. This is because the status record used to perform this check did not exist prior to V1R3. The z/OS1.3 system will format the status record, issue ILR029I as an informational message and continue to use the data set (along with the other system).

Once the IPL is complete, the status record is validated on a regular interval. If concurrent use of a data set is detected, both systems will be terminated with a 025 wait state code. Catalog information will also be validated with the status record. If the data set is deleted or key catalog information changes, the system will be terminated with a 025 wait state code.

Space Calculation Examples

Table 2-1 shows the values for page data sets. The examples following these figures show how to apply their tabular information to typical initialization considerations.

Note: After the system is running, you can use RMF reports to determine the sizes of page data sets. RMF reports provide the minimum, maximum, and average number of slots in use for page data sets. Thus, you can use the reports to adjust data set sizes, as needed.

Table 2-1. Page data set Values

Device Type	Slots/Cyl	Cyl/Meg
3380	150	1.7
3390	180	1.42
9345	150	1.7

Example 1: Sizing the PLPA Page Data Set, Size of the PLPA and Extended PLPA Unknown

Define the PLPA page data set to hold four megabytes; if that amount of space is exceeded, the remainder can be placed on the common page data set until the PLPA value is determined exactly.

Therefore: From the tables, 7 cylinders on a 3380 are needed for the 4-megabyte PLPA page data set. For the 3390, 6 cylinders are needed for the 4-megabyte PLPA page data set.

Example 2: Sizing the PLPA Page Data Set, Size of the PLPA and Extended PLPA Known

Assume the PLPA size is known to be 10 megabytes. Define the PLPA page data set to hold 10 megabytes plus 5%, or 10.5 megabytes. (The extra 5% allows for loss of space as a result of permanent I/O errors.)

Therefore: From the tables, 18 cylinders on a 3380 are needed for the 10.5-megabyte PLPA page data set. For the 3390, 15 cylinders are needed for the 10.5-megabyte PLPA page data set.

Example 3: Sizing the Common Page Data Set

Use the size of the virtual common service area (CSA) and extended CSA, defined by the CSA= parameter in the IEASYSxx parmlib member, as the minimum size for the common page data set. If the system is IPLed with a specification of CSA=(3000,80000), then the total CSA specified is 83000 kilobytes (approximately 81 megabytes). However, CSA and extended CSA always end on a segment boundary, so the system may round the size up by as much as 1023 kilobytes each. That rounding could make the CSA size as large as 83 megabytes with the CSA=(3000,80000) specification. After the system is running, you can use RMF reports to determine how much of the common page data set is being used and adjust the size of the data set accordingly.

Therefore: From the tables, 118 cylinders on a 3390 are needed to start with an 83-megabyte common page data set.

Example 4: Sizing Local Page Data Sets

Assume that the master scheduler address space and JES address space can each use about eight megabytes of private area storage. Next, determine the number of address spaces that will be used for subsystem programs such as VTAM and the system component address spaces, and allow eight megabytes of private area storage for each. To determine the amount of space necessary for batch address spaces, multiply the maximum number of batch address spaces that will be allowed to be active at once by the average size of a private area (calculated by the installation or approximated at eight megabytes).

To determine the amount of space necessary for TSO/E, multiply the maximum number of TSO/E address spaces allowed on the system at once by the average size of a private area (calculated by the installation or approximated at eight megabytes).

Next, determine the amount of space required for any large swappable applications which run concurrently. Use the allocated region size for the calculation.

Finally, estimate the space requirements for VIO data sets. Approximate this requirement by multiplying the expected number of VIO data sets used by the entire system by the average size of a VIO data set for the installation. After the system is fully loaded, you can use RMF reports to evaluate the estimates.

For example purposes, assume that the total space necessary for local page data sets is:

- 8 megabytes for the master scheduler address space
- 8 megabytes for the PC/AUTH address space
- 8 megabytes for the system trace address space

8	megabytes for the global resource serialization address space
8	megabytes for the allocation address space
8	megabytes for the communications task address space
8	megabytes for the dumping services address space
8	megabytes for the system management facilities address space
8	megabytes for the VTAM address space
8	megabytes for the JES address space
8	megabytes for the JES3AUX address space if JES3 is used
10	megabytes for the batch address space (10 batches x 1 megabyte each)
50	megabytes for TSO/E address spaces (50 TSO/E users x 1 megabyte each)
102	megabytes for large swappable application
+ 40	megabytes for VIO data sets (200 data sets x 0.2 megabyte each)
290	megabytes total + 15 meg (approx. 5%) buffer = 305 megabytes

Therefore: From the tables, 549 cylinders on 3380 type devices are necessary. For the 3390, 434 cylinders are necessary.

Note: Even when the local page data sets will fit on one 3390, you should spread them across more than one device. See performance recommendation number 5. If large swappable jobs or large VIO users are started and there is insufficient allocation of space on local page data sets, a system wait X'03C' could result.

The installation should also consider the extent of the use of data-in-virtual when calculating paging data set requirements. Users of data-in virtual may use sizable amounts of virtual storage which may put additional requirements on paging data sets.

The calculations shown here will provide enough local page space for the system to run. However, if a program continually requests virtual storage until the available local page data set space is constrained, this will not be enough space to prevent an auxiliary storage shortage.

Auxiliary storage shortages can cause severe performance degradation, and if all local page data set space is exhausted the system might fail. To avoid this, you can do one or more of the following:

- Use the IEFUSI user exit to set REGION limits for most address spaces and data spaces. If you have sufficient DASD, add enough extra local page data set space to accommodate one address space or data space of the size you allow in addition to the local page data set space the system usually requires, multiply the sum by 1.43, and allocate that amount of local page data set space. For more information about IEFUSI, see *z/OS MVS Installation Exits*.
- Overallocate local page data set space by 300% if you have sufficient DASD. This is enough auxiliary storage to prevent the system from reaching the auxiliary storage shortage threshold when the maximum amount of storage is obtained in a single address space or data space.

Performance Recommendations

The following recommendations might improve system performance through the careful use of paging data sets and devices:

1. Allocate only one paging data set per device. Doing this reduces contention among more than one data set for the use of the device.

Reason: If there is more than one paging data set on the same device, a significant seek penalty will be incurred. Additionally, if the data sets are local page data sets, placing more than one on a device can cause the data sets to be selected less frequently to fulfill write requests.

More than one paging data set on the same device also cause the loss of all gains in performance resulting from ASM using the suspend/resume function.

Comments: You might, however, place low-activity non-paging data sets on a device that holds a page data set and check for device contention by executing RMF to obtain direct access device activity reports during various time intervals. The RMF data on device activity count, percent busy, and average queue length should suggest whether device contention is a problem. RMF data for devices that contain only a page data set can be used as a comparison base.

2. Over-specify space for all page data sets.

Reason: Over-specifying space allows for the creation of additional address spaces before the deletion of current ones and permits some reasonable increase in the number of concurrent VIO data sets which may be backed by auxiliary storage. VIO data set growth might become a problem because there is no simple way to limit the total number of VIO data sets used by multiple jobs and TSO/E sessions. VIO data set paging can be controlled by restricting it to certain page data sets through the use of directed VIO.

VIO data set pages can be purged by a relPL specifying CVIO (or CLPA). CVIO indicates that the system is to ignore any VIO pages that exist on the page data sets and treat the page data sets initially as if there is no valid data on them (that is, there are no allocated slots). Thus, specifying CVIO prevents the warm start of jobs that use VIO data sets because the VIO pages have been purged. (For additional space considerations, see the guideline for estimating the total size of paging data sets in “Estimating Total Size of Paging Data Sets” on page 2-7.)

In all cases, ASM avoids scattering requests across large, over-specified, page data sets by concentrating its activity to a subset of the space allocated.

3. Use more than one local page data set, each on a unique device, even if the total required space is containable on one device.

Reason: When ASM uses more than one data set, it can page concurrently on more than one device. This is especially important during peak loads.

4. Distribute ASM data sets among channel paths and control units.

Reason: Although ASM attempts to use more than one data set concurrently, the request remains in the channel subsystem queues if the channel path or control unit is busy.

5. Dedicate channel paths and control units to paging devices.

Reason: In heavy paging environments, ASM can use the path to the paging devices exclusively for page-ins and page-outs and avoid interference with other users, such as IMS.

6. Make a page data set the only data set on the device.

Reason: Making a paging data set the only data set on the device enables ASM to effectively use the suspend/resume function. ASM can monopolize the device to its best performance advantage by controlling its own I/O processing of that data set. ASM doesn't have to perform the additional processing it would otherwise have to perform if I/O for any other data set, especially another page data set, were on the same device. If another data set must be placed on the device, select a low-use data set to minimize contention with the page data set.

7. Do not share volumes that contain page data sets among multiple systems.

Reason: While page data sets may be defined on volumes that contain shared non-paging data sets, they cannot be shared between systems.

8. Take one of the actions below to control the risk of auxiliary storage shortages. Auxiliary storage shortages have severe effects on system performance when they occur, and can also cause the system to fail. When a shortage occurs, the system rejects LOGON, MOUNT, and START commands and keeps address spaces with rapidly increasing auxiliary storage requirements from running until the shortage is relieved.
 - a. Use the SMF Step Initiation Exit, IEFUSI, to limit the sizes of most address spaces and data spaces.
 - b. If you do not establish limits using IEFUSI, consider overallocating local pagespace by an amount sufficient to allow a single address space or data space to reach the virtual storage limit (as might happen if a program looped obtaining storage) without exhausting virtual storage shortage.

See “Example 4: Sizing Local Page Data Sets” on page 2-4 for more information about calculating local page data set space requirements.

Estimating Total Size of Paging Data Sets

You can obtain a general estimate of the total size of all paging data sets by considering the following space factors.

1. The space needed for the common areas of virtual storage (PLPA and extended PLPA, MLPA, and CSA).
2. The space needed for areas of virtual storage: private areas of concurrent address spaces, and concurrently existing VIO page data sets. (The system portion of concurrent address spaces needs to be calculated only once, because it represents the same system modules.)

Using Measurement Facilities

You can possibly simplify the space estimate for the private areas mentioned above by picking an arbitrary value. Set up this amount of paging space, and then run the system with some typical job loads. To determine the accuracy of your estimate, start RMF while the jobs are executing. The paging activity report of the measurement program gives data on the number of unused 4K slots, the number of VIO data set pages backed by auxiliary storage, address space pages, and the number of unavailable (defective) slots.

The RMF report also contains the average values based on a number of samples taken during the report interval. These average values are in a portion of the report entitled “Local Page data set Slot Counts”. They are somewhat more representative of actual slot use because slot use is likely to vary during the report interval. The values from the paging activity report should enable you to adjust your original space estimate as necessary.

Adding More Paging Space

To add more paging space, you must use the DEFINE PAGESPACE command of Access Method Services to pre-format and catalog each new page data set. To add the page data set, you can use the PAGEADD operator command, or specify the data set at the next IPL on the PAGE parameter.

For More Information...

- See *z/OS DFSMS Access Method Services for Catalogs* for information about the DEFINE PAGESPACE command, and on the related commands - ALTER and DELETE - used for the handling of VSAM data sets.
- See *z/OS MVS System Commands* for a description of the PAGEADD command.
- See the description of the PAGE parameter in parmlib member IEASYSxx in *z/OS MVS Initialization and Tuning Reference*.

Deleting, Replacing or Draining Page Data Sets

You might need to remove a local page data set from the system for any of the following reasons:

- The hardware is being reconfigured.
- The hardware is generating I/O errors.
- The configuration of the page data set is being changed.
- System tuning requires the change.

The PAGEDEL command allows you to delete, replace or drain local page data set without an IPL. See *z/OS MVS System Commands* for the description of the PAGEDEL command.

ASM will reject a PAGEDEL command that will decrease the amount of auxiliary storage below an acceptable limit. ASM determines what is acceptable by examining SRM's auxiliary storage threshold constant, MCCASMTI. If it is determined that too much storage would be deleted by the PAGEDEL command, ASM will fail the page delete request.

Questions and Answers

Customers have often asked the following questions about ASM:

Q: *Does ASM use I/O load balancing?*

A: Yes, ASM does its own I/O load balancing.

When selecting a local page data set to fulfill a write request, ASM attempts to avoid overloading page data sets. ASM also attempts to favor those devices or channel paths that are providing the best service.

Q: *How does the auxiliary storage shortage prevention algorithm in SRM prevent shortages?*

A: It does so by swapping out address spaces that are accumulating paging space at a rapid rate. Page space is not immediately freed, but another job or TSO/E session (still executing) will eventually complete and free page space. SRM also prevents the creation of new address spaces and informs the operator of the shortage so that he can optionally cancel a job.

Q: *Is running out of auxiliary storage (paging space) catastrophic?*

A: No, not necessarily; it might be possible to add more page data sets with the PAGEADD operator command, optionally specifying the NONVIO system parameter. It may be necessary to reIPL to specify an additional

pre-formatted and cataloged page data set. (See the description of the PAGE parameter of the IEASYSxx member in *z/OS MVS Initialization and Tuning Reference*.)

Q: *Can we dynamically allocate more paging space?*

A: Yes. Additional paging space may be added with the PAGEADD operator command if the PAGTOTL parameter allowed for expansion (see the description of the PAGTOTL parameter of the IEASYSxx member in *z/OS MVS Initialization and Tuning Reference* and the PAGEADD command in *z/OS MVS System Commands*).

Q: *Can we remove paging space from system use?*

A: Yes. Use the PAGEDEL command for local page data sets.

Q: *How does ASM select slots?*

A: How ASM selects slots when writing out pages to page data sets depends on whether the write request is an individual request or a group request. For an individual write request, such as a request to write stolen pages (those pages changed since they were last read from the page data set), ASM selects any available slots. For a group write request, such as a request that results from a swap-out or a VIO move-out of groups of pages to page data sets, ASM attempts to select available slots that are contiguous. ASM also attempts to avoid scattering requests across large page data sets.

Q: *How does ASM select a local page data set for a page-out?*

A: ASM selects a local page data set for page-out from its available page data sets. ASM selects these data sets in a circular order within each type of data set, subject to the availability of free space and the device response time.

Q: *How will expanded storage affect my paging configuration?*

A: Expanded storage helps reduce the paging load to auxiliary storage devices. Therefore, you might be able to reduce the paging configuration below what was previously required when no expanded storage was configured.

Q: *How will z/Architecture affect my paging configuration?*

A: Expanded storage is not supported in z/Architecture. You can still use local page data sets in z/Architecture; however, because you can have more real storage in z/Architecture, less paging is likely to be needed.

Q: *Will data-in-virtual users increase the need for paging data sets?*

A: Data-in-virtual does provide applications with functions that would encourage extensive use of virtual storage. Depending on the extent of the usage of data-in-virtual, paging data set requirements may increase.

Chapter 3. The System Resources Manager

Important

Beginning with z/OS V1R3, WLM compatibility mode is no longer available. Accordingly, the IEAICSxx member, the IEAIPSxx member are no longer valid. Options in the IEAOPTxx member that were valid only in compatibility mode also are no longer valid. However, there are some parameters in IEAOPTxx that are still valid and still used by SRM in goal mode. For information on compatibility mode, see a previous version of this publication.

See *z/OS MVS Planning: Workload Management* for more information on WLM goal mode.

To a large degree, an installation's control over the performance of the system is exercised through the system resources manager (SRM).

"Section 1: Description of the System Resources Manager (SRM)" on page 3-2 discusses the types of control available through SRM, the functions used to implement these controls, and the concepts inherent in the use of SRM parameters. The parameters themselves are described in *z/OS MVS Initialization and Tuning Reference*.

"Section 2: Basic SRM Parameter Concepts" on page 3-11 discusses some basic OPT parameters. *z/OS MVS Initialization and Tuning Reference* provides descriptions of the OPT parameters and syntax rules.

"Section 3: Advanced SRM Parameter Concepts" on page 3-13 discusses some more advanced topics.

"Section 4: Guidelines" on page 3-16 provides some guidelines for defining installation requirements and preparing an initial OPT.

"Section 5: Installation Management Controls" on page 3-25 contains information about commands for SRM-related functions.

System Tuning and SRM

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

SRM, 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 SRM parameters might have to be adjusted to reflect changed conditions.

Section 1: Description of the System Resources Manager (SRM)

SRM is a component of the system control program. It determines which address spaces, of all active address spaces, should be given access to system resources and the rate at which each address space is allowed to consume these resources.

Before an installation turns to SRM, it should be aware of the response time and throughput requirements for the various types of work that will be performed on its system. Questions similar to the following should be considered:

- How important is turnaround time for batch work, and are there distinct types of batch work with differing turnaround requirements?
- Should subsystems such as IMS and CICS be controlled at all, or should they receive as much service as they request? That is, should they be allowed unlimited access to resources without regard to the impact this would have on other types of work?
- What is acceptable TSO/E response time for various types of commands?
- What is acceptable response time for compiles, sorts, or other batch-like work executed from a terminal?

Guidelines for defining installation requirements are discussed in “Section 4: Guidelines” on page 3-16.

Once these questions have been answered and, whenever possible, quantified, and the installation is reasonably confident that its requirements do not exceed the physical capacity of its hardware, it should then turn to SRM to specify the desired degree of control.

Controlling SRM

You can control the system resources manager (SRM) through the workload manager. In releases earlier than z/OS V1R3, an installation controlled SRM either through the IEAIPsxx and IEAICSxx parmlib members or through the workload manager. Controlling SRM through parmlib members was called workload management compatibility mode. Controlling SRM through the workload manager is called goal mode. Some parameters in the IEAOPTxx parmlib member applied only to compatibility mode (and are no longer valid), and some apply to goal mode.

With workload manager, you specify performance goals for work, and SRM adapts the system resources to meet the goals. SRM uses the same controls that exist today, but it sets them all dynamically based on the goals. For information on how to use workload manager, see *z/OS MVS Planning: Workload Management*.

While most information about how to use workload manager is in *z/OS MVS Planning: Workload Management*, many of the concepts that SRM uses dynamically in goal mode are explained in this publication, including the IEAOPTxx parameters.

Objectives

SRM bases its decision on two fundamental objectives:

1. To distribute system resources among individual address spaces in accordance with the installation's response, turnaround, and work priority requirements.
2. To achieve optimal use of system resources as seen from the viewpoint of system throughput.

SRM attempts to ensure optimal use of system resources by periodically monitoring and balancing resource utilization. If resources are under-utilized, SRM will attempt to increase the system load. If resources are over-utilized, SRM will attempt to reduce the system load.

Types of Control

SRM offers three distinct types of control to an installation:

- Service classes
- Dispatching control, for all address spaces
- Period

SRM sets the values of controls dynamically based on the performance goals for work defined in a service policy. The remainder of this section describes these types of controls and the functions that SRM uses to implement them.

Dispatching Control

Dispatching priorities control the rate at which address spaces are allowed to consume resources after they have been given access to these resources. This form of competition takes place outside the sphere of domain control, that is, all address spaces compete with all other address spaces with regard to dispatching priorities.

Functions

This topic discusses the functions used by SRM to implement the controls described in “Dispatching Control.”

The functions are as follows:

1. Swapping (see “Swapping”)
2. Dispatching of work (see “Dispatching of Work” on page 3-5)
3. Resource use functions (see “Resource Use Functions” on page 3-5)
4. Enqueue delay minimization (see “Enqueue Delay Minimization” on page 3-7)
5. I/O priority queueing (see “I/O Priority Queueing” on page 3-7)
6. DASD device allocation (see “DASD Device Allocation” on page 3-8)
7. Prevention of storage shortages (see “Prevention of Storage Shortages” on page 3-8)
8. Pageable frame stealing (see “Pageable Frame Stealing” on page 3-10).

Swapping

Swapping is the primary function used by SRM to exercise control over distribution of resources and system throughput. Using system status information that is periodically monitored, SRM determines which address spaces should have access to system resources.

In addition to the swapping controls described in the following text, SRM also provides an optional swap-in delay to limit the response time of TSO/E transactions.

There are several reasons for swapping. Some swaps are used for control of domains and the competition for resources between individual address spaces within a domain, while others provide control over system-wide performance and help increase the throughput.

Domain-Related Swaps:

- *Unilateral swap in:* If the number of a domain's address spaces that are in the multiprogramming set (MPS) is less than the number SRM has set for the swap-in target, SRM swaps in additional address spaces for that domain, if possible.
- *Unilateral swap out:* If the number of a domain's address spaces that are in the multiprogramming set is greater than the number SRM has set for the swap-out target, SRM swaps out address spaces from that domain.
- *Exchange swap:* All address spaces of a domain compete with one another for system resources. When an address space in the multiprogramming set ¹ has exceeded its allotted portion of resources, relative to an address space of the same domain waiting to be swapped in, SRM performs an exchange swap. That is, the address space in the multiprogramming set is swapped out and the other address space is swapped in. This competition between address spaces is described in detail in "Section 2: Basic SRM Parameter Concepts" on page 3-11.

System-Related Swaps:

- *Swaps due to storage shortages:* Two types of shortages cause swaps: auxiliary storage shortages and pageable frame shortages. If the number of available auxiliary storage slots is low, SRM will swap out the address space that is acquiring auxiliary storage at the fastest rate. For a shortage of pageable frames, if the number of fixed frames is very high, SRM will swap out the address space that acquired the greatest number of fixed frames. This process continues until the number of available slots rises above a fixed target, or until the number of fixed frames falls below a fixed target.
- *Swaps to improve central storage usage:* The system will swap out an address space when the system determines that the current mix of address spaces is not best utilizing central storage. The system swaps out address spaces to create a positive effect on system paging and swap costs.
- *Swap out an address space to make room for an address space:* The system will swap in an address space when the system determines that it has been out longer than its recommendation value would dictate. See "Working Set Management" on page 3-6 for information about the recommendation value.
- *Swaps due to wait states:* In certain cases, such as a batch job going into a long wait state (LONG option specified on the WAIT SVC, an STIMER wait specification of greater than or equal to 0.5 seconds, an ENQ for a resource held by a swapped out user), the address space will itself signal SRM to be swapped out in order to release storage for the use of other address spaces. Another example would be a time sharing user's address space that is waiting for input from the terminal after a transaction has completed processing. SRM also detects address spaces in a wait state. That is, address spaces in central storage that are not executable for a fixed interval will be swapped. (See "Logical Swapping" on page 3-6.)
- *Request Swap:* The system may request that an address space be swapped out. For example, the CONFIG STOR, OFFLINE command requests the swap out of address spaces that occupy frames in the storage unit to be taken offline.
- *Transition Swap:* A transition swap occurs when the status of an address space changes from swappable to nonswappable. For example, the system performs a transition swap out before a nonswappable program or V=R step gets control. This special swap prevents the job step from improperly using reconfigurable storage.

1. The multiprogramming set consists of those address spaces that are in central storage and are eligible for access to the processor.

Swap Recommendation Value

SRM calculates a swap recommendation value to determine which address spaces to use in an exchange or unilateral swap. A high swap recommendation value indicates that the address space is more likely to be swapped in. As the swap recommendation value decreases, that address space is more likely to be swapped out.

The swap recommendation value for an address space that is swapped in ranges from 100 to -999 as service is accumulated. An address space must accumulate enough CPU service to justify the cost of a swap out. Once the swap recommendation value goes below 0, the address space is ready to be swapped out in an exchange swap.

The swap recommendation value for an address space that is swapped out and ready to come in ranges from 0 to 998 as the address space remains out. Once the swap recommendation value goes above 100, the address space has been out long enough to justify the cost of the exchange swap.

For an address space that is swapped out but not ready to come in, the swap recommendation value as reported by the RMF Monitor II ASD report is meaningless. The swap recommendation value for an address space that is out too long is reported as 999. Also, if an address space has been assigned long-term storage protection (as described in the “Storage Protection” section of the “Workload Management Participants” chapter in *z/OS MVS Planning: Workload Management*), then the swap recommendation value is 999.

For monitored address spaces, SRM calculates a working set manager recommendation value. See “Working Set Management” on page 3-6 for information about the working set manager recommendation value.

Dispatching of Work

Dispatching of work is done on a priority basis. That is, the ready work with the highest priority is dispatched first. The total range of priorities is from 191 to 255.

Note: Certain system address spaces execute at the highest priority and are exempt from installation prioritization decisions.

Resource Use Functions

The resource use functions of SRM attempt to optimize the use of system resources on a system-wide basis, rather than on an individual address space basis. The functions are as follows:

- Logical swapping - SRM automatically performs logical swapping when sufficient central storage is available.
- Working set management - SRM automatically determines the best mix of work in the multiprogramming set (MPS) and the most productive amount of central storage to allocate to each address space.

Multiprogramming Level Adjusting: SRM monitors system-wide utilization of resources, such as the CPU and paging subsystem, and seeks to alleviate imbalances, that is, over-utilization or under-utilization. This is accomplished by periodically adjusting the number of address spaces that are allowed in central storage and ready to be dispatched for appropriate domains (multiprogramming set).

When system contention factors indicate that the system is not being fully utilized, SRM will select a domain and increase the number of address spaces allowed access to the processor for that domain, thereby increasing utilization of the system.

Logical Swapping: To use central storage more effectively and reduce processor and channel subsystem overhead, the SRM logical swap function attempts to prevent the automatic physical swapping of address spaces. Unlike a physically swapped address space, where the LSQA, fixed frames, and recently referenced frames are placed on auxiliary storage, SRM keeps the frames that belong to a logically swapped address space in central storage.

Address spaces swapped for wait states (for example, TSO/E terminal waits) are eligible to be logically swapped out whenever the think time associated with the address space is less than the system threshold value. SRM adjusts this threshold value according to the demand for central storage. SRM uses the unreferenced interval count (UIC) to measure this demand for central storage. As the demand for central storage increases, SRM reduces the system threshold value; as the demand decreases, SRM increases the system threshold value.

The system threshold value for think time fluctuates between low and high boundary values. The installation can change these boundary values in the IEAOPTxx parmlib member. The installation can also set threshold values for the UIC; setting these threshold values affects how SRM measures the demand for central storage.

SRM logically swaps address spaces if the real frames they own are not required to immediately replenish the supply of available real frames. Any address space subject to a swap out can become logically swapped as long as there is enough room in the system. Address spaces with pending request swaps or those marked as swaps due to storage shortages are the only address spaces that are physically swapped immediately.

Large address spaces that have been selected to be swapped to replenish central storage are trimmed before they are swapped. The trimming is done in stages and only to the degree necessary for the address space to be swapped. In some cases, it might be necessary to trim pages that have been recently referenced in order to reduce the address space to a swappable size.

SRM's logical swapping function periodically checks all logically swapped out address spaces to determine how long they've been logically swapped out. SRM physically swaps out those address spaces that have been logically swapped out for a period greater than the system threshold value for think time only when it is necessary to replenish the supply of available frames.

Working Set Management: SRM automatically determines the best mix of work in the multiprogramming set (MPS) and the most productive amount of central storage to allocate to each address space within MPL constraints.

To achieve this, SRM monitors the system paging, page movement, and swapping rates and productive CPU service for all address spaces to detect when the system might run more efficiently with selected address space working sets managed individually. If the system is spending a significant amount of resources for paging, SRM will start monitoring the central storage of selected address spaces.

After SRM decides that an address space should be monitored, SRM collects additional address space data. Based on this data, SRM might discontinue implicit block paging.

For monitored address spaces, SRM calculates a working set manager recommendation value that can override the swap recommendation value. See “Swap Recommendation Value” on page 3-5 for information about the swap recommendation value. The working set manager recommendation value measures the value of adding the address space to the current mix of work in the system. Even when the swap recommendation value indicates that a specific address space should be swapped in next, the working set manager recommendation value might indicate that the address space should be bypassed. To ensure that no address space is repeatedly bypassed, the system swaps in a TSO/E user 30 seconds after being bypassed. For all other types of address spaces, the system will swap in the address space 10 minutes after being bypassed.

If a monitored address space is paging heavily, SRM might manage its central storage usage. When an address space is managed, SRM imposes a central storage target (implicit dynamic central storage isolation maximum working set) on an address space.

Enqueue Delay Minimization

This function deals with the treatment of address spaces enqueued upon system resources that are in demand by other address spaces or resources for which a RESERVE has been issued and the device is shared. If an address space controlling an enqueued resource is swapped out and that resource is required by another address space, SRM will ensure that the holder of the resource is swapped in again as soon as possible.

Once in central storage, a swap out of the controlling address space would increase the duration of the enqueue bottleneck. Therefore, the controlling address space is given a period of CPU service during which it will not be swapped due to service considerations (discussed in “Section 2: Basic SRM Parameter Concepts” on page 3-11.) The length of this period is specified by the installation by means of a tuning parameter called the enqueue residence value (ERV), contained in parmllib member IEAOPTxx.

I/O Priority Queueing

I/O priority queueing is used to control deferred I/O requests. If this function is invoked, all deferred I/O requests, except paging and swapping, will be queued according to the I/O priorities associated with the requesting address spaces. Paging and swapping are always handled at the highest priority. An address space's I/O priority is by default the same as its dispatching priority. All address spaces in one mean-time-to-wait group fall into one I/O priority. In addition, address spaces that are time sliced have their I/O queued at their time slice priority. Changes to an address space's dispatching priority when the address space is time sliced up or down, do not affect the I/O priority.

An installation can assign an I/O priority that is higher or lower than the dispatching priority for selected groups of work. For example, if the installation is satisfied with the dispatching priority of an interactive application but would like the application's I/O requests to be processed before those of other address spaces executing at the same priority, the application could be given an I/O priority higher than its dispatching priority.

If I/O priority queueing is not invoked, all I/O requests are handled in a first-in/first-out (FIFO) manner.

DASD Device Allocation

Device allocation selects the most responsive DASD devices as candidates for permanent data sets on mountable devices (JCL specifies nonspecific VOLUME information or a specific volume and the volume is not mounted).

The ability of SRM to control DASD device allocation is limited by the decision an installation makes at system installation time and at initial program loading (IPL) time, as well as by the user's JCL parameters. SRM can only apply its selection rules to a set of DASD devices that are equally acceptable for scheduler allocation. This set of devices does not necessarily include all the DASD devices placed in an esoteric group during system installation. At that time, an esoteric group is defined by the UNITNAME macro and entered in the eligible device table (EDT). During system installation each esoteric group is partitioned into subgroups if either of the following conditions occurs:

- The group includes DASD devices that are common with another esoteric group.
- The group includes DASD devices that have certain generic differences. System installation partitions only esoteric groups that consist of magnetic tape or direct access devices.

For example, assume that you specify the following at system installation time:

```
UNITNAME=DASD,UNIT=((470,7),(478,8),(580,6))  
UNITNAME=SYSDA,UNIT=((580,6))
```

Because of the intersection with SYSDA (580,6), the DASD group is divided into two subgroups: (470,7) and (478,8) in one subgroup and (580,6) in the other.

Allocation allows SRM to select from only one subgroup at a time. After allocating all devices in the first subgroup, allocation selects DASD devices from the next subgroup. Using the previous example, when a job requests UNIT=DASD, allocation tells SRM to select a device from the first group (470-476 and 478-47F) regardless of the relative use of channel paths 4 and 5. After all of the DASD devices in the first group have been allocated, allocation tells SRM to select devices from the second group (580-585).

Prevention of Storage Shortages

SRM periodically monitors the availability of three types of storage and attempts to prevent shortages from becoming critical. The three types of storage are:

- Auxiliary storage
- SQA
- Pageable frames.

Auxiliary storage: When more than a fixed percentage (constant MCCASMT1) of auxiliary storage slots have been allocated, SRM reduces demand for this resource by taking the following steps:

- LOGON, MOUNT and START commands are inhibited until the shortage is alleviated.
- Initiators are prevented from executing new jobs.
- The target MPL (both in and out targets) in each domain is set to its minimum value.
- The operator is informed of the shortage.

- Choosing from a subset of swappable address spaces, SRM stops the address space(s) acquiring slots at the fastest rate and prepares the address space for swap-out (logical swap). When SRM swaps-out an address space because of excessive slot usage, SRM informs the operator of the name of the job that is swapped out, permitting the operator to cancel the job.

If the percentage of auxiliary slots allocated continues to increase (constant MCCASMT2), SRM informs the operator that a critical shortage exists. SRM then prevents all unilateral swap-ins (except for domain zero). This action allows the operator to cancel jobs or add auxiliary paging space to alleviate the problem.

When the shortage has been alleviated, the operator is informed and SRM halts its efforts to reduce the demand for auxiliary storage.

SQA: When the number of available SQA and CSA pages falls below a threshold, SRM:

- Inhibits LOGON, MOUNT, and START commands until the shortage is alleviated.
- Informs the operator that an SQA shortage exists.

If the number of available SQA and CSA pages continues to decrease, SRM informs the operator that a critical shortage of SQA space exists and, except for domain zero, SRM prevents all unilateral swap-ins.

When the shortage has been alleviated, the operator is informed and SRM halts its efforts to prevent acquisition of SQA space.

Pageable Frames: SRM attempts to ensure that enough pageable central storage is available to the system. SRM monitors the amount of pageable storage available, ensures that the currently available pageable storage is greater than a threshold, and takes continuous preventive action from the time it detects a shortage of pageable storage until the shortage is relieved.

When SRM detects a shortage of pageable frames caused by an excess of fixed or DREF storage, SRM uses event code ENVPC055 to signal an ENF event. When the shortage is relieved, SRM signals another ENVPC055 event to notify listeners that the shortage is relieved. SRM does not raise the signal for the “shortage relieved” condition until a delay of 30 seconds following the most recent occurrence of a fixed-storage shortage. The intent of signalling this event is to give system components and subsystems that use fixed or DREF storage an opportunity to help relieve the shortage.

Regardless of the cause of a shortage of pageable storage, SRM takes these actions:

- Inhibits LOGON, MOUNT, and START commands until the shortage is relieved
- Prevents initiators from executing new jobs
- Informs the operator that a shortage of pageable storage exists

Further SRM actions to relieve the shortage depend on the particular cause of the shortage.

The following system conditions can cause a shortage of pageable storage:

- Too many address spaces are already in storage.
Too many address spaces in storage does not usually, of itself, cause a shortage of pageable storage because SRM performs MPL adjustment and logical swap threshold adjustment, which generally keep an adequate amount of fixed storage available to back the address spaces.

- Too much page fixing is taking place.

One or more address spaces are using substantial amounts of storage either through explicit requests to fixed virtual storage or by obtaining virtual storage that is page fixed by attributes such as LSQA or SQA.

There are different types of pageable storage shortages:

- A shortage of pageable storage below 16 megabytes.
- A shortage when pageable storage has reached a threshold.
- A shortage when fixed and DREF allocated to CASTOUT=NO ESO Hiperspaces has reached a threshold.

If too much page fixing is the cause of the shortage of pageable storage, SRM:

1. Identifies the largest swappable user or users of fixed storage. If any of these users own more frames than three times the median fixed frame count, SRM begins to physically swap them out, starting with the user with the largest number of fixed pages. SRM continues to swap users out until it releases enough fixed storage to relieve the shortage.
2. Begins to physically swap out the logically-swapped out address spaces if swapping out the largest users of fixed storage does not relieve the shortage of pageable storage.
3. Decreases the MPLs for those domains that have the lowest contention indices if physically swapping out the logically-swapped out address spaces does not relieve the shortage of pageable storage. This MPL adjustment allows the swap analysis function of SRM to swap out enough address spaces to relieve the shortage.

SRM takes the following additional actions if the shortage of pageable storage reaches a critical threshold:

1. Informs the operator that there is a critical shortage.
2. Repeats all the steps described above that are applicable to the cause of the shortage.
3. Prevents any unilateral swap-in, except for domain zero.

When the shortage of pageable storage is relieved, SRM waits for a delay of 30 seconds following the most recent occurrence of a fixed shortage, and then:

- Allows new address spaces to be created through the LOGON, MOUNT, and START commands.
- Notifies the operator that the shortage is relieved.
- Allows initiators to process new jobs.
- Allows unilateral swap-ins.

Note: Those address spaces that SRM swapped out to relieve the shortage of pageable storage are not swapped back in if their storage requirements would potentially cause another shortage to occur.

Pageable Frame Stealing

Pageable frame stealing is the process of taking an assigned central storage frame away from an address space to make it available for other purposes, such as to satisfy a page fault or swap in an address space.

When there is a demand for pageable frames, SRM will steal those frames that have gone unreferenced for a long time and return them to the system. The unreferenced interval count (UIC) represents the time in seconds for a complete

steal cycle. A complete steal cycle is the time the stealing routine needs to check all frames in the system. When there is a demand for storage, the stealing routine:

- tests the reference bit of a frame
- decides whether to steal the frame
- schedules the page-out.

When there is no demand for storage, no stealing occurs.

The UIC algorithm forecasts the UIC, based on the current stealing rate. The UIC can vary between 0 and 65535 and gets calculated every second. When there is no demand for storage in the system (no stealing occurs) the system has a UIC of 65535. If there is a very high demand for storage in the system, the system has a UIC close to 0.

Stealing takes place strictly on a demand basis, that is, there is no periodic stealing of long-unreferenced frames. A complete steal cycle can take days.

SRM modifies the stealing process for address spaces that it is managing and for address spaces that are storage critical. For these address spaces, SRM attempts to enforce the address space's real storage target that was set when SRM decided that the address space was to be managed.

I/O Service Units

The number of *I/O service units* is a measurement of individual data set I/O activity and JES spool reads and writes for all data sets associated with an address space. SRM calculates I/O service using I/O block (EXCP) counts.

When an address space executes in cross-memory mode (that is, during either secondary addressing mode or a cross-memory call), the EXCP counts are included in the I/O service total. This I/O service is not counted for the address space that is the target of the cross-memory reference.

Section 2: Basic SRM Parameter Concepts

This section discusses the OPT parameters for these basic SRM specifications:

- MPL adjustment control
- Transaction definition for CLISTs
- Directed VIO activity
- Alternate wait management

For explanations of the OPT parameters, see *z/OS MVS Initialization and Tuning Reference*.

See “Section 3: Advanced SRM Parameter Concepts” on page 3-13 for information about advanced OPT concepts.

MPL Adjustment Control

The OPT provides keywords to specify upper and lower thresholds for the variables that SRM uses to determine if it should increase, decrease, or leave the MPL unchanged. When one of these variables exceeds its threshold value, SRM regards this change as a signal to adjust the MPL. The following chart gives the internal names for the control variables, their thresholds, and conditions that can influence a change.

Control variable and internal name	Thresholds that can influence an MPL change:		Keyword in
	decrease	increase	OPT
CPU utilization (RCVCPUA)	>RCCCPUTH	<RCCCPUTL	RCCCPUT
Page fault rate (RCVPTR)	>RCCPTRTH	<RCCPTRTL	RCCPTRT ¹
UIC (RCVUICA)	<RCCUICTL	>RCCUICTH	RCCUICT
Percentage of online storage fixed (RCVFXIOP)	>RCCFXTTH	<RCCFXTTL	RCCFXTT
Percentage of storage that is fixed within the first 16 megabytes (RCVMFXA)	>RCCFXETH	<RCCFXETL	RCCFXET

¹The default thresholds for this keyword causes the corresponding control variable to have no effect on MPL adjustment.

Transaction Definition for CLISTs

An installation can specify whether the individual commands in a TSO/E CLIST are treated as separate TSO/E commands for transaction control. Specifying CNTCLIST=YES causes a new transaction to be started for each command in the CLIST. A possible exposure of specifying CNTCLIST=YES is that long CLISTs composed of trivial and intermediate commands might monopolize a domain's MPL slots and cause interactive terminal users to be delayed. Specifying CNTCLIST=NO (the default) causes the entire CLIST to constitute a single transaction.

Directed VIO Activity

VIO data set pages can be directed to a subset of the local paging data sets through *directed VIO*, which allows the installation to direct VIO activity away from selected local paging data sets that will be used only for non-VIO paging. With directed VIO, faster paging devices can be reserved for paging where good response time is important. The NONVIO system parameter, with the PAGE system parameter, allows the installation to define those local paging data sets that are not to be used for VIO, leaving the rest available for VIO activity. However, if space is depleted on the paging data sets made available for VIO paging, the non-VIO paging data sets will be used for VIO paging.

The installation uses the DVIO keyword to either activate or deactivate directed VIO.

Note: The NONVIO and PAGE system parameters are in the IEASYSxx parmlib member.

Alternate Wait Management

An installation can specify whether to activate or deactivate alternate wait management (AWM). If AWM is activated, SRM and LPAR cooperate to reduce low utilization effects and overhead.

Specifying CCCAWMT with any value in the range 1 to 499999 makes AWM active. Specifying CCCAWMT with any value in the range 500000 to 1000000 makes AWM inactive. The default is 12000 (AWM active).

Dispatching Mode Control

An installation can switch between the HiperDispatch mode enabled or HiperDispatch mode disabled by specifying the HIPERDISPATCH keyword for the parmlib member IEAOPT. The default is NO (HiperDispatch mode disabled).

In vertical mode, units of work can be preferably dispatched on affinity nodes. If the total physical processor equivalent of the defined weight of a partition is less than 1.5 standard processor CPUs, a console message is issued as a warning that the system is not configured as recommended. zAAP and zIIP processors will not be considered when deciding the weight of a partition.

Section 3: Advanced SRM Parameter Concepts

This section includes information about selective enablement for I/O and adjustment of constants options.

Selective Enablement for I/O

Selective enablement for I/O is a function that SRM uses to control the number of processors that are enabled for I/O interruptions. The intent of this function is to enable only the minimum number of processors needed to handle the I/O interruption activity without the system incurring excessive delays. That is, if one processor can process the I/O interruptions without excessive delays, then only one processor need be enabled for I/O interruptions.

At system initialization, one processor is enabled for I/O interruptions. To determine if a change should be made to the number of processors that are enabled, SRM periodically monitors I/O interruptions.

By comparing this value to threshold values, SRM determines if another processor should be enabled or if an enabled processor should be disabled for I/O interruptions. If the computed value exceeds the upper threshold, I/O interruptions are being delayed, and another processor (if available) will be enabled for I/O interruptions. If the value is less than the lower threshold (and more than one processor is enabled), a processor will be disabled for I/O interruptions. The installation can change the threshold values using the CPENABLE parameter in the IEAOPTxx parmlib member.

A processor that enters a wait state is always enabled for I/O interruptions, however, regardless of what you specify for the CPENABLE keyword.

In addition to enabling a processor when I/O activity requires it, SRM also enables another processor for I/O interruptions if one of the following occurs:

- An enabled processor is taken offline.
- An enabled processor has a hardware failure.
- SRM detects that no I/O interruptions have been taken for a predetermined period of time and concludes that the enabled processor is unable to accept interrupts.

An installation can use the CPENABLE keyword to specify low and high thresholds for the percentage of I/O interruptions to be processed through the test pending

interrupt (TPI) instruction. SRM uses these thresholds to determine if a change should be made to the number of processors enabled for I/O interruptions.

The following chart gives the internal names of the control variables and indicates their relation to the condition.

Control variable and internal name	Percentage of I/O Interruptions		Keyword in OPT
	under	over	
Percentage of I/O interruptions through TPI instruction (ICVTPIP)	<ICCTPILO	>ICCTPIHI	CPENABLE

Table 3-1 relates SRM seconds to real time. The SRM constants that are shown in this table are merely generalizations and approximations. For more accurate comparisons of processors, see the internal throughput rate (ITR) numbers in *Large Systems Performance Reference (LSPR)*, SC28-1187.

Table 3-1. Relating SRM Seconds to Real Time

Processor Model	SRM Seconds/Real Seconds
Processors: zSeries® 990	
zSeries 990 Models 301 - 332	508.1298
Processors: zSeries 900	
zSeries 900 Models 101–109	269.3964
zSeries 900 Models 110–116, 1C1–1C9	281.5314
Processors: zSeries 890	
zSeries 890 Models 110, 210, 310, 410	29.4117
zSeries 890 Models 120, 220, 320, 420	52.0399
zSeries 890 Models 130, 230, 330, 430	99.5222
zSeries 890 Models 140, 240, 340, 440	124.2544
zSeries 890 Models 150, 250, 350, 450	194.0993
zSeries 890 Models 160, 260, 360, 460	236.7423
zSeries 890 Models 170, 270, 370, 470	413.9071
Processors: zSeries 800	
zSeries 800 Model 0E1	45.4545
zSeries 800 Model 0A1	90.8430
zSeries 800 Model 0X2	98.5804
zSeries 800 Model 0B1	130.4801
zSeries 800 Model 0C1	162.3376
zSeries 800 Model 0A2	160.2563
zSeries 800 Model 001 - 004	217.0138
Processors: S/390® 9672 G6 Models	
S/390 Models 9672-X17 - 9672-XZ7	194.0993
S/390 Models 9672-Z17 - 9672-ZZ7	224.8200
Processors: S/390 9672 G5 Models	
S/390 Model 9672-R16	129.9376
S/390 Model 9672-R26	129.9376
S/390 Models 9672-R36 - 9672-R96	141.0834
S/390 Models 9672-RA6, 9672-RB6	97.9623
S/390 Models 9672-RC6, 9672-RD6	129.9376
S/390 Models 9672-RX6, 9672-T16, 9672-T26	141.0834
S/390 Models 9672-Y16 - 9672-YX6	168.4635
Processors: S/390 3000	
S/390 3000 Model A10	41.6666

Table 3-1. Relating SRM Seconds to Real Time (continued)

Processor Model	SRM Seconds/Real Seconds
S/390 3000 Model A20	39.9872
Processors: S/390 2003	
S/390 2003 Model 107	27.8520
S/390 2003 Model 124 (All Models)	30.3988 (per CPU)
S/390 2003 Model 1C5 (All Models)	37.6279 (per CPU)
S/390 2003 Model 2X7 (All Models)	46.0914 (per CPU)
S/390 2003 Model 203	6.1814
S/390 2003 Model 204	10.3203
S/390 2003 Model 205	14.4375
S/390 2003 Model 206	18.5680
S/390 2003 Model 207	27.8520
S/390 2003 Model 215	33.3333
S/390 2003 Model 216	41.6666
S/390 2003 Model 224 (All Models)	30.5325 (per CPU)
S/390 2003 Model 225 (All Models)	39.9872 (per CPU)
S/390 2003 Model 246 (All Models)	41.1455 (per CPU)
S/390 2003 Model 2C5 (All Models)	37.6279 (per CPU)

Adjustment of Constants Options

Certain OPT parameters make it more convenient for installations with unique resource management requirements to change some SRM constants. The defaults provided are adequate for most installations. A parameter needs to be specified only when its default is found to be unsuitable for a particular system environment. The following functions can be modified by parameters in the OPT:

- Enqueue residence control
- SRM invocation interval control
- Pageable storage control
- Central storage control

Enqueue residence control

This parameter, specified by the ERV keyword, defines the amount of CPU service that the address space is allowed to receive before it is considered for a workload recommendation swap out. The parameter applies to all swapped-in address spaces that are enqueued on a resource needed by another user. For more information, see “Enqueue Delay Minimization” on page 3-7.

SRM invocation interval control

This parameter, specified by the RMPTTOM keyword, controls the invocation interval for SRM timed algorithms. Increasing this parameter above the default value reduces the overhead caused by SRM algorithms, such as swap analysis and time slicing. However, when these algorithms are invoked at a rate less than the default, the accuracy of the data on which SRM decisions are made, and thus the decisions themselves, might be affected.

Pageable storage control

Two keywords are provided in the OPT to signal a shortage of pageable storage. Keyword MCCFXTPR specifies the percentage of storage that is fixed. Keyword MCCFXEPR specifies the percentage of storage, within the first 16 MB, that needs to be fixed before SRM detects a shortage.

Control variable and internal name	Shortage of pageable storage exists	Keyword in OPT
Percentage of storage that is fixed RCETOTFX ¹	>MCCFXTTPR	MCCFXTTPR
Percentage of storage that is fixed within the first 16 megabytes (RCEBELFX) ¹	>MCCFXEPR	MCCFXEPR
¹ These variables are actual frame counts rather than percentages. SRM multiplies the MCCFXTTPR threshold by the amount of online storage and multiplies the MCCFXEPR threshold by the amount of storage eligible for fixing in order to arrive at the threshold frame counts that it uses to compare against the actual frame counts. If MCCFXEPR x (amount of storage eligible for fixing) is greater than MCCFXTTPR x (amount of online storage), then the threshold frame counts that SRM uses to compare against the actual frame counts are set equal.		

Central Storage Control

This parameter, specified by the MCCAFC TH keyword, indicates the number of frames on the available frame queue when stealing begins and ends. The range of values on this keyword determines the block size that SRM uses for stealing. In order to get a block into central storage, the lower value of the range must be greater than the block size.

Section 4: Guidelines

This section provides some guidelines for these tasks:

- Defining installation requirements and objectives
- Preparing the initial OPT

Defining Installation Requirements

Before specifying any parameters to SRM, an installation must define response and throughput requirements for its various classification of work.

Examples of specific questions that should be answered are listed below. The applicability of these questions will, of course, vary from installation to installation.

Subsystems

How many subsystems will be active at any one time and what are they?

For IMS, how many active regions will there be?

Will the subsystem address space(s) be nonswappable?

What is the desired response time and how will it be measured?

Batch

What is the required batch throughput or turnaround for various job classes?

How much service do batch jobs require, and what service rate is needed to meet the turnaround requirement?

An RMF workload report or reduction of SMF data in type 5 or type 30 records will provide the average service consumed by jobs of different classes. Based on service definition coefficients of CPU=10.0,IOC=5.0,MSO=3.0,SRB=10.0; the following approximations can be made:

Short jobs use 30,000 service units or less.

Long jobs use more than 30,000 units.

What is the average number of ready jobs?

Most likely, this is the number of active initiators. A few extra initiators may be started to decrease turnaround times.

TSO/E

What is the number of terminals?

What is the average number of ready users?

As a guideline for installations new to TSO/E, assume that an installation doing program development on 3270 terminals will have two ready users for every ten users logged on. This average will vary, depending on the type of terminal and on the type of TSO/E session (data entry, problem solving, program development).

What is the required response time and expected transaction rate for different categories of TSO/E transactions at different times, such as peak hours?

What is the expected response time for short transactions?

How will this response time be measured?

Should response time be different for select groups of TSO/E users?

How should semi-trivial and non-trivial transactions be treated?

How are they defined?

An installation can use RMF workload reports or SMF data in type 34 and 35 records available to help define trivial and non-trivial TSO/E work. Based on service definition coefficients of CPU=10.0,IOC=5.0,MSO=3.0,SRB=10.0; the following approximations can be made:

Short TSO/E commands use 200 service units or less.

Medium length commands use between 200 and 1000 service units.

Long TSO/E commands use 1000 service units or more.

What is the required service rate for TSO/E users?

If 2-second response time (as reported by RMF) is required for very short TSO/E commands (100 service units), the required service rate for such a transaction is $100/2$ or 50 service units per second. Service rates for other types of transactions should be computed also.

General

What is the importance level of TSO/E, batch, IMS, and special batch classes in relation to one another?

Which may be delayed or “tuned down” to satisfy other requirements?

In other words, which response requirements are fixed and which are variable?

What percentage of system resources should each group receive?

Preparing an Initial OPT

There are several approaches to preparing an initial OPT.

- Use the default OPT.

- Modify the default OPT.
- Create a new OPT.

The following tables describe the service consumed per second of execution time by CPU model. The values listed are SRM constants. The total system absorption rate reported by RMF will not equal the values listed here because these do not include certain types of system processing.

Table 3-2. IBM System z9 Enterprise Class (z9 EC) Processor Models

System z9 EC	Service Units Per Second of Task or SRB Execution Time	Seconds Task or SRB Execution Time Per Service Unit
Processor Model		
z9 EC, Model 701	29520.2952	0.000034
z9 EC, Model 702	28368.7943	0.000035
z9 EC, Model 703	27538.7263	0.000036
z9 EC, Model 704	26845.6376	0.000037
z9 EC, Model 705	26143.7908	0.000038
z9 EC, Model 706	25559.1054	0.000039
z9 EC, Model 707	25000.0000	0.000040
z9 EC, Model 708	24427.4809	0.000041
z9 EC, Model 709	23845.0075	0.000042
z9 EC, Model 710	23391.8129	0.000043
z9 EC, Model 711	22922.6361	0.000044
z9 EC, Model 712	22566.9958	0.000044
z9 EC, Model 713	22099.4475	0.000045
z9 EC, Model 714	21739.1304	0.000046
z9 EC, Model 715	21390.3743	0.000047
z9 EC, Model 716	21052.6316	0.000048
z9 EC, Model 717	20833.3333	0.000048
z9 EC, Model 718	20592.0206	0.000049
z9 EC, Model 719	20356.2341	0.000049
z9 EC, Model 720	20125.7862	0.000050
z9 EC, Model 721	19900.4975	0.000050
z9 EC, Model 722	19777.5031	0.000051
z9 EC, Model 723	19656.0197	0.000051
z9 EC, Model 724	19536.0195	0.000051
z9 EC, Model 725	19417.4757	0.000052
z9 EC, Model 726	19300.3619	0.000052
z9 EC, Model 727	19070.3218	0.000052
z9 EC, Model 728	18845.7008	0.000053
z9 EC, Model 729	18735.3630	0.000053
z9 EC, Model 730	18626.3097	0.000054
z9 EC, Model 731	18518.5185	0.000054
z9 EC, Model 732	18411.9678	0.000054

Table 3-2. IBM System z9 Enterprise Class (z9 EC) Processor Models (continued)

System z9 EC	Service Units Per Second of Task or SRB Execution Time	Seconds Task or SRB Execution Time Per Service Unit
Processor Model		
z9 EC, Model 733	18285.7143	0.000055
z9 EC, Model 734	18161.1805	0.000055
z9 EC, Model 735	18161.1805	0.000055
z9 EC, Model 736	18038.3315	0.000055
z9 EC, Model 737	17917.1333	0.000056
z9 EC, Model 738	17797.5528	0.000056
z9 EC, Model 739	17679.5580	0.000057
z9 EC, Model 740	17563.1175	0.000057
z9 EC, Model 741	17448.2007	0.000057
z9 EC, Model 742	17448.2007	0.000057
z9 EC, Model 743	17334.7779	0.000058
z9 EC, Model 744	17222.8202	0.000058
z9 EC, Model 745	17112.2995	0.000058
z9 EC, Model 746	17003.1881	0.000059
z9 EC, Model 747	16895.4593	0.000059
z9 EC, Model 748	16895.4593	0.000059
z9 EC, Model 749	16771.4885	0.000060
z9 EC, Model 750	16649.3236	0.000060
z9 EC, Model 751	16528.9256	0.000061
z9 EC, Model 752	16410.2564	0.000061
z9 EC, Model 753	16293.2790	0.000061
z9 EC, Model 754	16177.9575	0.000062

Table 3-3. zSeries 990 Processor Models

zSeries 990	Service Units Per Second of Task or SRB Execution Time	Seconds Task or SRB Execution Time Per Service Unit
Processor Model		
zSeries 990, Model 301	21857.9	0.000046
zSeries 990, Model 302	20752.3	0.000048
zSeries 990, Model 303	20075.3	0.000050
zSeries 990, Model 304	19559.9	0.000051
zSeries 990, Model 305	19047.6	0.000053
zSeries 990, Model 306	18626.3	0.000054
zSeries 990, Model 307	18202.5	0.000055
zSeries 990, Model 308	17777.8	0.000056
zSeries 990, Model 309	17353.6	0.000058
zSeries 990, Model 310	17003.2	0.000059
zSeries 990, Model 311	16666.7	0.000060
zSeries 990, Model 312	16326.5	0.000061

Table 3-3. zSeries 990 Processor Models (continued)

zSeries 990 Processor Model	Service Units Per Second of Task or SRB Execution Time	Seconds Task or SRB Execution Time Per Service Unit
zSeries 990, Model 313	15984.0	0.000063
zSeries 990, Model 314	15640.3	0.000064
zSeries 990, Model 315	15296.4	0.000065
zSeries 990, Model 316	14953.3	0.000067
zSeries 990, Model 317	14787.4	0.000068
zSeries 990, Model 318	14611.9	0.000068
zSeries 990, Model 319	14532.2	0.000069
zSeries 990, Model 320	14440.4	0.000069
zSeries 990, Model 321	14349.8	0.000070
zSeries 990, Model 322	14260.2	0.000070
zSeries 990, Model 323	14171.8	0.000071
zSeries 990, Model 324	14084.5	0.000071
zSeries 990, Model 325	13998.3	0.000071
zSeries 990, Model 326	13913.0	0.000072
zSeries 990, Model 327	13828.9	0.000072
zSeries 990, Model 328	13745.7	0.000073
zSeries 990, Model 329	13663.5	0.000073
zSeries 990, Model 330	13582.3	0.000074
zSeries 990, Model 331	13490.7	0.000074
zSeries 990, Model 332	13400.3	0.000075

Table 3-4. zSeries 900 Processor Models

zSeries 900 Processor Model	Service Units Per Second of Task or SRB Execution Time	Seconds Task or SRB Execution Time Per Service Unit
zSeries 900, Model 101	11585.8	0.000086
zSeries 900, Model 102	10891.8	0.000092
zSeries 900, Model 103	10430.2	0.000096
zSeries 900, Model 104	10081.9	0.000099
zSeries 900, Model 105	9732.4	0.000103
zSeries 900, Model 106	9384.2	0.000107
zSeries 900, Model 107	9153.3	0.000109
zSeries 900, Model 108	8805.7	0.000114
zSeries 900, Model 109	8456.7	0.000118
zSeries 900, Model 110	9334.9	0.000107
zSeries 900, Model 111	9211.3	0.000109
zSeries 900, Model 112	8968.6	0.000112

Table 3-4. zSeries 900 Processor Models (continued)

zSeries 900 Processor Model	Service Units Per Second of Task or SRB Execution Time	Seconds Task or SRB Execution Time Per Service Unit
zSeries 900, Model 113	8724.1	0.000115
zSeries 900, Model 114	8602.2	0.000116
zSeries 900, Model 115	8359.5	0.00012
zSeries 900, Model 116	8117.7	0.000123
zSeries 900, Model 1C1	12112	0.000083
zSeries 900, Model 1C2	11502.5	0.000087
zSeries 900, Model 1C3	11142.1	0.00009
zSeries 900, Model 1C4	10781.7	0.000093
zSeries 900, Model 1C5	10540.2	0.000095
zSeries 900, Model 1C6	10296	0.000097
zSeries 900, Model 1C7	10056.6	0.000099
zSeries 900, Model 1C8	9816	0.000102
zSeries 900, Model 1C9	9575.1	0.000104
zSeries 900, Model 210	11165.3873	0.000090
zSeries 900, Model 211	10869.5652	0.000092
zSeries 900, Model 212	10723.8606	0.000093
zSeries 900, Model 213	10430.2477	0.000096
zSeries 900, Model 214	10139.4170	0.000099
zSeries 900, Model 215	9993.7539	0.000100
zSeries 900, Model 216	9696.9697	0.000103
zSeries 900, Model 2C1	14692.3783	0.000068
zSeries 900, Model 2C2	13961.6056	0.000072
zSeries 900, Model 2C3	13377.9264	0.000075
zSeries 900, Model 2C4	13082.5838	0.000076
zSeries 900, Model 2C5	12638.2306	0.000079
zSeries 900, Model 2C6	12345.6790	0.000081
zSeries 900, Model 2C7	12048.1928	0.000083
zSeries 900, Model 2C8	11756.0617	0.000085
zSeries 900, Model 2C9	11461.3181	0.000087

Table 3-5. zSeries 890 Processor Models

zSeries 800	Service Units Per Second of	Seconds Task or SRB
Processor Model	Task or SRB Execution	Execution Time Per Service
	Time	Unit
zSeries 890, Model 110	1264.9221	0.000791
zSeries 890, Model 210	1225.2087	0.000816
zSeries 890, Model 310	1200.3901	0.000833
zSeries 890, Model 410	1175.6062	0.000851
zSeries 890, Model 120	2238.0753	0.000447
zSeries 890, Model 220	2167.7280	0.000461
zSeries 890, Model 320	2123.7059	0.000471
zSeries 890, Model 420	2079.8128	0.000481
zSeries 890, Model 130	4280.3638	0.000234
zSeries 890, Model 230	4146.1519	0.000241
zSeries 890, Model 330	4061.9447	0.000246
zSeries 890, Model 430	3978.1203	0.000251
zSeries 890, Model 140	5344.0214	0.000187
zSeries 890, Model 240	5176.3183	0.000193
zSeries 890, Model 340	5071.3154	0.000197
zSeries 890, Model 440	4965.8597	0.000201
zSeries 890, Model 150	8346.3745	0.000120
zSeries 890, Model 250	8084.8914	0.000124
zSeries 890, Model 350	7920.7921	0.000126
zSeries 890, Model 450	7755.6956	0.000129
zSeries 890, Model 160	10184.5958	0.000098
zSeries 890, Model 260	9864.3650	0.000101
zSeries 890, Model 360	9661.8357	0.000104
zSeries 890, Model 460	9461.8569	0.000106
zSeries 890, Model 170	17797.5528	0.000056
zSeries 890, Model 270	17241.3793	0.000058
zSeries 890, Model 370	16895.4593	0.000059
zSeries 890, Model 470	16546.0186	0.000060

Table 3-6. zSeries 800 Processor Models

zSeries 800	Service Units Per Second of	Seconds Task or SRB
Processor Model	Task or SRB Execution	Execution Time Per Service
	Time	Unit
zSeries 800, Model 0E1	1955.0	0.000512
zSeries 800, Model 0A1	3907.2	0.000256
zSeries 800, Model 0X2 UNI	4239.5	0.000236
zSeries 800, Model 0X2	3900.5	0.000256
zSeries 800, Model 0B1	5612.0	0.000178
zSeries 800, Model 0C1	6980.8	0.000143

Table 3-6. zSeries 800 Processor Models (continued)

zSeries 800	Service Units Per Second of Task or SRB Execution Time	Seconds Task or SRB Execution Time Per Service Unit
Processor Model		
zSeries 800, Model 0A2 UNI	6893.6	0.000145
zSeries 800, Model 0A2	6341.7	0.000158
zSeries 800, Model 001	9334.9	0.000107
zSeries 800, Model 002	8588.3	0.000116
zSeries 800, Model 003	8121.8	0.000123
zSeries 800, Model 004	7843.1	0.000128

Table 3-7. S/390 9672 Processor Models

S/390 9672	Service Units Per Second of Task or SRB Execution Time	Seconds of Task or SRB Execution Time Per Service Unit
Processor Model		
S/390 9672, Model T16	6067.5	0.000165
S/390 9672, Model T26	5643.7	0.000177
S/390 9672, Model R36	5460.8	0.000183
S/390 9672, Model R46	5278.8	0.000189
S/390 9672, Model R56	5158.0	0.000194
S/390 9672, Model R66	5036.2	0.000199
S/390 9672, Model R76	4915.5	0.000203
S/390 9672, Model R86	4733.7	0.000211
S/390 9672, Model R96	4490.6	0.000223
S/390 9672, Model RX6	4247.4	0.000235
S/390 9672, Model Y16	7246.4	0.000138
S/390 9672, Model Y26	6739.7	0.000148
S/390 9672, Model Y36	6449.0	0.000155
S/390 9672, Model Y46	6230.5	0.000161
S/390 9672, Model Y56	6086.0	0.000164
S/390 9672, Model Y66	5941.3	0.000168
S/390 9672, Model Y76	5797.1	0.000173
S/390 9672, Model Y86	5578.8	0.000179
S/390 9672, Model Y96	5361.9	0.000187
S/390 9672, Model YX6	5071.3	0.000197
S/390 9672, Model RA6	4212.7	0.000237
S/390 9672, Model RB6	3960.4	0.000253
S/390 9672, Model R16	5588.5	0.000179
S/390 9672, Model R26	5141.4	0.000194

Table 3-7. S/390 9672 Processor Models (continued)

S/390 9672 Processor Model	Service Units Per Second of Task or SRB Execution Time	Seconds of Task or SRB Execution Time Per Service Unit
S/390 9672, Model RC6	5029.9	0.000199
S/390 9672, Model RD6	4918.5	0.000203
S/390 9672 Processor Model		
S/390 9672, Model X17	8346.4	0.000120
S/390 9672, Model X27	7928.6	0.000126
S/390 9672, Model X37	7677.5	0.000130
S/390 9672, Model X47	7511.7	0.000133
S/390 9672, Model X57	7262.8	0.000138
S/390 9672, Model X67	7095.3	0.000141
S/390 9672, Model X77	6762.5	0.000148
S/390 9672, Model X87	6512.0	0.000154
S/390 9672, Model X97	6177.6	0.000162
S/390 9672, Model XX7	6010.5	0.000166
S/390 9672, Model XY7	5759.5	0.000174
S/390 9672, Model XZ7	5592.5	0.000179
S/390 9672, Model Z17	9667.7	0.000103
S/390 9672, Model Z27	9184.8	0.000109
S/390 9672, Model Z37	8893.8	0.000112
S/390 9672, Model Z47	8700.4	0.000115
S/390 9672, Model Z57	8412.2	0.000119
S/390 9672, Model Z67	8217.8	0.000122
S/390 9672, Model Z77	7831.6	0.000128
S/390 9672, Model Z87	7540.1	0.000133
S/390 9672, Model Z97	7249.7	0.000138
S/390 9672, Model ZX7	6959.5	0.000144
S/390 9672, Model ZY7	6765.3	0.000148
S/390 9672, Model ZZ7	6475.1	0.000154

Table 3-8. S/390 3000 processor models

S/390 3000 Processor Model	Service Units Per Second of Task or SRB Execution Time	Seconds of Task or SRB Execution Time Per Service Unit
S/390 3000, Model A10	1792.1	0.000558
S/390 3000, Model A20	1582.3	0.000632

If you plan to use these constants for purposes other than those suggested in this section, observe the following limitations:

- Actual customer workloads and performance may vary. For a more exact comparison of processors, see the internal throughput rate (ITR) numbers in *Large Systems Performance Reference (LSPR)*.
- CPU time can vary for different runs of the same job step. One or more of the following factors might cause variations in the CPU time: CPU architecture (such as storage buffering), cycle stealing with integrated channels, and the amount of the queue searching (see *z/OS MVS System Management Facilities (SMF)*).
- The constants are applied as if each logical partition was an independent machine. The constants do not account for multiprocessor effects that might be due to the number of processors shared by other logical partitions. SRM uses the same set of hardware provided service unit factors in all partitions; the number of logical general purpose processors determine the correct factor. For example, a logical 1-way partition on a z9 EC, Model 701 is rated at 29,520 service units per second, while a 10-way partition on the same machine is rated at 23,392 service units per second. The logical 1-way to 10-way ratio implied by the service unit definition is affected by the fact that all logical processors experience a hardware multiprocessor effect more like a 10-way than a 1-way. The one shared logical partition in the 1-way partition will not execute at the same rate as a single dedicated logical partition on a z9 EC, Model 701.

Using SMF Task Time

For installations with no prior service data, the task time reported in SMF record Type 4, 5, 30, 34, and 35 records can be converted to service units using the preceding tables.

Section 5: Installation Management Controls

This section contains information about commands for SRM-related installation management functions.

Operator Commands Related to SRM

The system operator can directly influence SRM's control of specific jobs or groups of jobs by entering commands from the console. The exact formats of these commands are defined in *z/OS MVS System Commands*.

The SET command with the OPT parameter is used to switch to a different OPT after an IPL. SRM bases all control decisions for existing and future jobs on the parameters in the new parmlib member.

Appendix. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

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

Notices

This information was developed for products and services offered in the USA.

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

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

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

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

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

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

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

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

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Programming Interface Information

This book is intended to help the customer initialize and tune the MVS element of z/OS. This book documents information that is NOT intended to be used as Programming Interfaces of z/OS.

Trademarks

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

- CICS
- DB2
- DFSMS
- DFSMS/MVS
- DFSMSdfp

- DFSMShsm
- DFSMSrmm
- ESCON
- IBM
- IBMLink
- IMS
- Language Environment
- RMF
- S/390
- SOM
- SOMobjects
- VTAM
- z/Architecture
- z/OS
- z9
- zSeries

UNIX is a registered trademark of The Open Group in the United States and other countries. Other company, product and service names may be the trademarks or service marks of others.

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

Index

A

- access time
 - for modules 1-18
- accessibility A-1
- address space
 - in virtual storage
 - description 1-10
 - layout in virtual storage 1-4, 1-11
 - swapping 3-3
 - system
 - creation 1-2
- algorithm
 - auxiliary storage shortage prevention 2-8
 - paging operations 2-1
 - slot allocation 2-1
- allocation
 - See also* device allocation, volume attribute list
 - considerations 1-42
 - device allocation 1-43
 - improving performance 1-43
 - virtual storage
 - considerations 1-12
- alternate wait management 3-13
- ASM (auxiliary storage manager)
 - expanded storage affects paging 2-9
 - I/O load balancing 2-8
 - initialization 2-1
 - local page data set selection 2-9
 - overview 1-30
 - page data set
 - effect on system performance 2-2
 - estimating total size 2-7
 - size 2-2
 - page operations 2-1
 - performance
 - recommendations 2-5
 - questions and answers 2-8
 - shortage prevention 2-8
 - z/Architecture affects paging 2-9

B

- batch
 - requirements 3-16

C

- central storage control
 - modifying 3-16
- central storage space
 - V=R region 1-9
- central storage usage
 - swaps used 3-4
- COFVLFxx member
 - coding 1-36
- command
 - SRM, relationship 3-25

- common area
 - in virtual storage 1-11
- common page data set 2-2
 - sizing 2-4
- common storage tracking function
 - tracking use of common storage 1-29
- compatibility mode 3-2
- constant
 - adjusting through IEAOPTxx 3-15
- CSA (common service area)
 - description 1-24
 - tracking use of common storage 1-29
 - using the common storage tracking function 1-29
- CSVLLAxx member
 - coding 1-35
 - specifying the FREEZEINOFREEZE option 1-39
 - specifying the GET_LIB_ENQ keyword 1-41
 - specifying the MEMBERS keyword 1-38
 - specifying the REMOVE keyword 1-39
- CVIO specification 2-6

D

- data set
 - page
 - size recommendations 2-2
 - paging
 - description 1-32
 - estimating total size 2-7
 - system data set
 - description 1-30
- data-in-virtual
 - page data set
 - affected by data-in-virtual 2-9
- demand swap 3-6
- device allocation
 - DASD
 - function used by SRM 3-8
- device selection
 - recommendations 2-5
- directed VIO 3-12
 - page data set 1-32
- disability A-1
- dispatching
 - controlled by SRM 3-3
 - priority 3-5
- Dispatching Mode Control 3-13
- dynamic allocation 1-42

E

- EDT (eligible device table) 1-43
- enqueue delay minimization
 - function used by SRM 3-7
- enqueue residence control
 - modifying 3-15
- exchange swap 3-4

- expanded storage
 - affecting page configuration 2-9
 - overview 1-10

F

- fixed LSQA
 - storage requirements 1-9
- FLPA (fixed link pack area)
 - description 1-8
- FREEZE/NOFREEZE option 1-39

G

- GET_LIB_ENQ keyword 1-41
- GETMAIN macro
 - macro request
 - variable-length 1-28
- GETMAIN/FREEMAIN/STORAGE (GFS) trace
 - tracing the use of storage macros 1-30
- goal mode 3-2

I

- I/O (input/output)
 - load balancing
 - by ASM 2-8
 - function supported by DASD device
 - allocation 3-8
 - selective enablement for I/O by SRM 3-13
 - storage space
 - virtual 1-33
- I/O priority
 - queueing function used by SRM 3-7
- I/O service units
 - definition 3-11
 - used by SRM 3-11
- IEALIMIT installation exit
 - description 1-28
- IEASYSxx parmlib member 1-44
- IEFUSI installation exit
 - description 1-28
- initialization
 - JES2 1-5
 - JES3 1-5
 - master scheduler 1-5
- installation requirements
 - defining 3-16
- IPL (initial program load)
 - major functions 1-1

J

- JES2 1-42
 - initialization 1-5
 - region size
 - limiting 1-28
- JES3
 - address space
 - auxiliary 1-5

- JES3 (*continued*)
 - initialization 1-5
 - region size
 - limiting 1-28
- JES3AUX address space 1-5
- job pack area
 - in system search order 1-15
- JOBLIB
 - in system search order 1-15

K

- keyboard A-1

L

- layout of virtual storage
 - single address space 1-11
- LLA (library lookaside)
 - CSVLLAxx member 1-35
 - LLACOPY macro 1-38
 - modification 1-38
 - MODIFY LLA command 1-38
 - modifying shared data sets 1-39
 - overview 1-33
 - planning to use 1-34
 - refreshing LLA-managed libraries 1-38
 - removing libraries from LLA management 1-39
 - START command 1-37
 - STOP LLA command 1-38
 - using the FREEZE/NOFREEZE option 1-39
 - using the GET_LIB_ENQ keyword 1-41
- LLACOPY macro 1-38
 - directory
 - modification 1-38
- load balancing
 - DASD device allocation 3-8
- local page data set
 - selection by ASM 2-9
 - sizing 2-4
- logical swapping
 - used by SRM 3-6
- LOGON command
 - processing 1-5
- LPA
 - in system search order 1-15
 - placement of modules 1-19
- LSQA (local system queue area)
 - description 1-25
 - fixed storage requirement 1-9
 - storage requirement
 - fixed 1-9

M

- map of virtual storage
 - address space 1-4
- master scheduler
 - initialization, description 1-5
- MEMBERS keyword
 - of CSVLLAxx 1-38

- MLPA (modified link pack area)
 - description 1-23
 - specification at IPL 1-23
- MODIFY LLA command 1-38
- module library 1-42
- module search order
 - description 1-14, 1-15, 1-34
- mount and use attribute for volume
 - assigning
 - using a VATLSTxx parmlib member 1-44
 - using the MOUNT command 1-44
- MOUNT command
 - processing 1-5
- MPL (multiprogramming level)
 - adjusting function used by SRM 3-5
- MPL adjustment control
 - modifying 3-11
- multiprogramming set 3-4

N

- NIP (nucleus initialization program)
 - major functions 1-1
- non-sharable attribute 1-46
- Notices B-1
- nucleus area
 - description 1-8

O

- OPT
 - initial parameter values
 - selecting 3-17
 - parameter on the SET command 3-25
 - preparing the initial 3-17
- out-of-space condition 1-28

P

- page data set
 - description 1-32
 - directed VIO 1-32
 - estimating size 2-7
 - measurement facilities 2-7
 - size
 - recommendations 2-2
 - space calculation
 - values 2-3
- page operation
 - algorithms 2-1
- page space
 - adding 2-7
- pageable frame stealing
 - used by SRM 3-10
- pageable storage control
 - modifying 3-15
- paging
 - space
 - adding 2-9
 - affected by expanded storage 2-9
 - affected by z/Architecture 2-9

- paging (*continued*)
 - space (*continued*)
 - depletion 2-8
 - effects of data-in-virtual 2-9
 - removing 2-9
- paging operation
 - algorithms 2-1
- performance
 - affected by storage placement 1-16, 1-19
 - recommendations 1-16, 1-19, 1-21
 - ASM (auxiliary storage manager) 2-5
- permanently resident volume 1-44
 - mount attribute 1-44
 - notes 1-45
 - use attributes
 - assigning 1-45
 - volumes that are always permanently resident 1-45
- PLPA (pageable link pack area)
 - data set 2-2
 - description 1-14
 - IEAPAKxx 1-14
 - primary and secondary 1-32
- priority
 - dispatching 3-5
- private area in virtual storage 1-11
- private area user region
 - description 1-26
 - real region 1-26
 - virtual region 1-26
- processor model
 - related to service units 3-18
 - related to SRM seconds 3-14
 - related to task/SRB execution time 3-18
- processor models
 - S/390 3000 3-24
 - S/390 9672 3-23
 - System z9 EC 3-18
 - zSeries 800 3-22
 - zSeries 890 3-22
 - zSeries 900 3-20
 - zSeries 990 3-19
- processor storage
 - overview 1-6

R

- real regions in private area user region 1-26
- real time
 - related to SRM seconds 3-14
- recommendation value
 - swap 3-5
 - working set manager 3-7
- region 1-27
 - size
 - limiting 1-27
- REGION parameter
 - on the JOB/EXEC statement 1-28
- REMOVE keyword
 - of CSVLLAxx 1-39
- request swap 3-4

resource use function
used by SRM 3-5

S

S/390 3000

processor models 3-24

S/390 9672

processor models 3-23

search order

for modules 1-14, 1-15, 1-34

selective enablement for I/O

by SRM 3-13

modifying 3-13

serialization

of devices during allocation 1-42

service unit

related to processor model 3-18

related to task/SRB execution time 3-18

SET command 3-25

shortcut keys A-1

slot

algorithm for allocating 2-1

ASM selection 2-9

space over-specification 2-6

SQA (system queue area)

fixed, description 1-9

storage shortage prevention 3-9

virtual, description 1-12

SRB (service request block)

execution time

related to processor model 3-18

related to service units 3-18

SRM (system resources manager)

and system tuning 3-1

constants 3-15

control, types 3-3

description 3-2

dispatching control 3-3

examples 3-16

functions used by 3-3

guidelines 3-16

installation control specification

concepts 3-11

installation management controls 3-25

installation requirements

batch 3-16

guidelines 3-17

subsystems 3-16

introduction 3-1

invocation interval control

modifying 3-15

IPS (installation performance specification)

concepts 3-11

objectives 3-2

operator commands related to SRM 3-25

OPT concepts 3-11

options 3-3

parameters

concepts 3-11

preparing initial OPT 3-17

SRM (system resources manager) *(continued)*

requirements

TSO/E 3-17

SRM seconds

based on processor model 3-14

related to real time 3-14

timing parameters 3-14

START command 1-42

processing 1-5

START LLA command 1-37

STEPLIB

in system search order 1-15

storage

auxiliary storage

overview 1-30

expanded storage

overview 1-10

processor storage

overview 1-6

virtual storage

address space 1-10

overview 1-10

storage initialization

ASM 2-1

storage management

initialization process 1-1

overview 1-1

storage shortage

prevention

for pageable frames 3-9

swaps result 3-4

storage shortage prevention

for auxiliary storage 3-8

for SQA 3-9

function used by SRM 3-8

subpools 229, 230, 249

description 1-25

subsystem

requirements 3-16

SWA (scheduler work area)

description 1-25

swap

causes

improve system paging rate 3-4

storage shortage 3-4

to improve central storage usage 3-4

demand 3-6

wait state 3-4

swap dataset

and virtual I/O storage space 1-33

swap recommendation value 3-5

swapping

logical

used by SRM 3-6

types 3-3

used by SRM 3-3

system address space

creation 1-2, 1-4

system data set

description 1-30

- system initialization
 - process 1-1
- system paging rate
 - swap used 3-4
- system preferred area
 - description 1-7
- system region
 - description 1-25
- system tuning
 - SRM discussion 3-1
- System z9 EC
 - processor models 3-18

T

- TASKLIB
 - in system search order 1-15
- transaction
 - definition
 - for CLIST 3-12
- transition swap 3-4
- TSO/E
 - requirements 3-17

U

- UIC (unreferenced interval count)
 - definition 3-10
- unilateral swap out 3-4
- unilateral swap-in 3-3
- user region 1-27

V

- V=R central storage space
 - description 1-9
- VATLSTxx parmlib member 1-43
- VIO (virtual input/output)
 - dataset page
 - directed 3-12
 - directed
 - paging data set 1-32
 - storage space 1-33
- virtual region
 - in private area user region 1-26
- virtual storage
 - affect of placement of modules 1-20
 - allocation
 - considerations 1-12
 - layout 1-11
 - map 1-11
 - overview 1-10
 - problem identification 1-29
 - single address space 1-11
 - tracing the use of storage macros 1-30
 - using GETMAIN/FREEMAIN/STORAGE (GFS)
 - trace 1-30
- virtual storage address space
 - description 1-10
- VLF (virtual lookaside facility)
 - starting 1-37

- VLF (virtual lookaside facility) *(continued)*
 - STOP VLF command 1-38
 - used with LLA 1-35
- volume attribute list 1-43

W

- wait state
 - swaps result 3-4
- wall clock time
 - related to SRM seconds 3-14
- working set management 3-6
- working set manager
 - recommendation value 3-7

Z

- z/Architecture
 - affecting page configuration 2-9
- zSeries 800
 - processor models 3-22
- zSeries 890
 - processor models 3-22
- zSeries 900
 - processor models 3-20
- zSeries 990
 - processor models 3-19

Readers' Comments — We'd Like to Hear from You

z/OS
MVS Initialization and Tuning Guide

Publication No. SA22-7591-06

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape



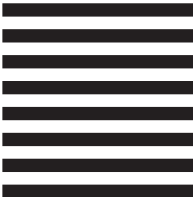
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5694-A01

Printed in USA

SA22-7591-06

