



---

## Session 21584 Diagnosing Problems in a z/OS UNIX Environment: Operational PD

z/OS Core Technologies – August 9<sup>th</sup>, 2017

John Shebey [jshebey@us.ibm.com](mailto:jshebey@us.ibm.com)  
IBM Poughkeepsie

**SHARE**  
EDUCATE • NETWORK • INFLUENCE



**SHARE**  
Providence 2017

**SHARE**  
EDUCATE • NETWORK • INFLUENCE  
August 6-11  
Rhode Island Convention Center  
Providence, RI



# Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- MVS
- OS/390®
- z/Architecture®
- z/OS®

\* Registered trademarks of IBM Corporation

For a complete list of IBM trademarks, see: <http://www.ibm.com/legal/us/en/copytrade.shtml>

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

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

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

\* All other products may be trademarks or registered trademarks of their respective companies.

#### Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

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

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

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

---



# Table of Contents

---

- z/OS UNIX Overview . . . . . 4
- Messages and Codes . . . . . 15
- Console Commands . . . . . 23
- PD Documentation . . . . . 33
- PD Procedures . . . . . 41
- Problem Examples . . . . . 48
- Reference Information. . . . . 64

Appendix

- Appendix A - Signal Numbers . . . . . 69
- Appendix B - OMVS CTRACE to External Writer . . . . . 73
- Appendix C - Formatting OMVS CTRACE . . . . . 75



# z/OS UNIX Overview



# What is z/OS UNIX?

- **z/OS UNIX System Services**
  - UNIX operating system that runs on z/OS MVS
  - AT&T created the original UNIX operating system in 1969, and in 1973, source code was made available outside of AT&T
    - Code was ported to other platforms with added extensions
  - In 1994, as part of MVS SP4.3, IBM introduced UNIX, calling it OpenEdition
  - z/OS UNIX conforms to the open standards of [POSIX \(Portable Operating System Interface for Computer Environments\)](#)
    - Interface standard governed by the IEEE and based on UNIX
    - Covers a wide spectrum of operating system components ranging from C language and shell interfaces to system administration
- BSD & System V were two other main UNIX branches

z/OS UNIX System Services (first introduced by IBM as OpenEdition in 1994) is a UNIX operating system that runs on z/OS MVS. UNIX history dates back to 1969 when AT&T created the original UNIX operating system. Prior to z/OS UNIX, the 2 main UNIX branches were System V (System 5) UNIX developed by AT&T, and BSD (Berkeley Software Distribution) UNIX developed by CSRG (Computer System Research Group) at UCB (University of California at Berkeley). The IEEE POSIX group implemented open UNIX standards and specifications, which z/OS UNIX conforms to. This aids with the porting of UNIX applications to z/OS.



# Basic Terminology

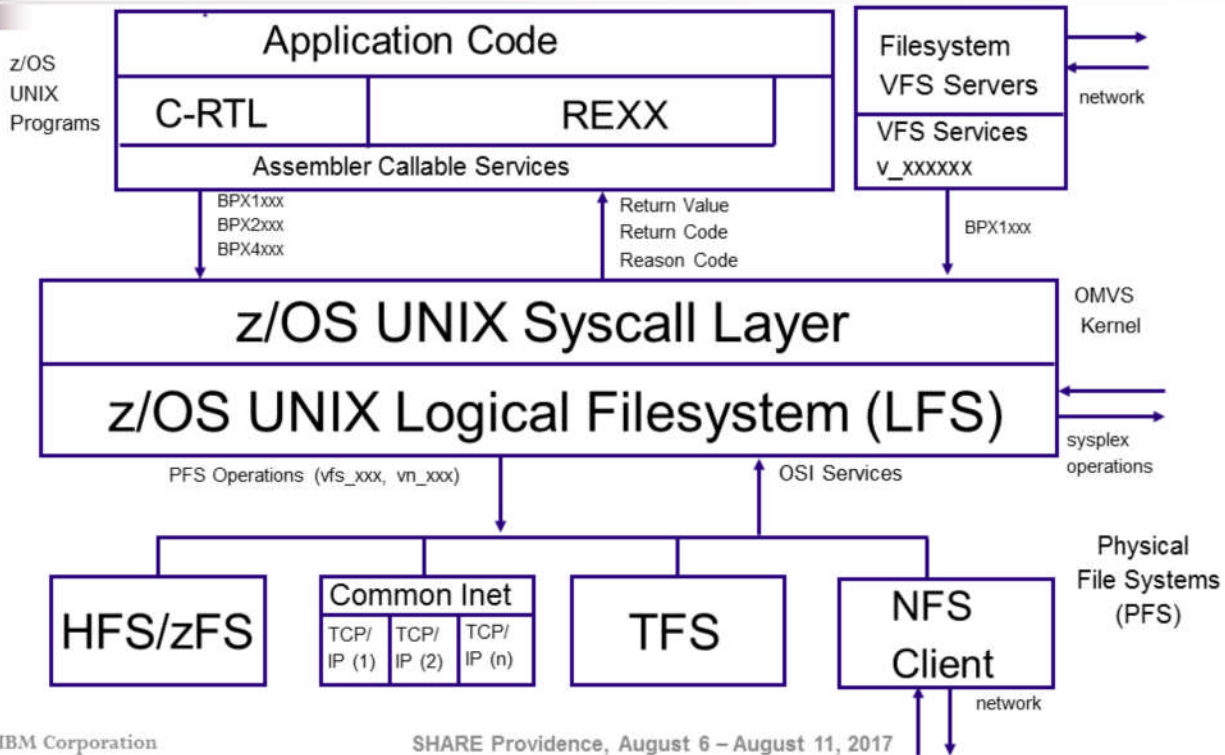
---

- **Process** – program using z/OS UNIX services
  - Generally, a process is at the address space level
    - However, there can be multiple processes in single address space (e.g. TCPIP)
- **Thread** – unit of work (task) in a process
- **Dub** – establish a z/OS UNIX environment for an address space and/or task
  - create a process and/or thread
- **Fork/Spawn** – methods by which a parent process creates a child process
- **Zombie** – address space with OMVS resources remaining after dubbed process terminates
- **Signal** – interrupt mechanism between processes

The OASB is the OMVS extension of the ASSB. If the AssbOasb is non-zero, then the address space contains at least 1 process.

The OTCB is the OMVS extension of the STCB. If the StcbOtcB is non-zero, then the TCB is a dubbed thread (known to z/OS UNIX).

# z/OS UNIX Environment



Programs using z/OS UNIX normally are written in high level languages such as C or REXX. The languages 'transform' into z/OS UNIX assembler callable services (syscalls) via the POSIX API. Assembler callable services can also be called directly.

Physical file data can reside on DASD (HFS/zFS) or be accessed through the network (sockets / NFS). Communication to Physical File Systems (PFS's) is through PFS operations - `vfs_ops` for operations on entire filesystems; `vn_ops` for operations on individual files. Operating System Interface (OSI) operations provide specific callable services to PFS's. VFS servers use the VFS callable services API (`v_XXXX`) to issue file requests (using FIDs rather than pathnames). There are other Physical File Systems (TFS, CINET, etc.) that are defined by FILESYSTYPE statements in BPXPRMxx.



# Shared File System Overview

- Ability to share file system data across sysplex

- Messaging protocol (via XCF services) used to transfer data in sysplex between owner and client systems

- Common File System Hierarchy for all systems in Couple Data Set (CDS) - type **BPXMCDS**

- Mount and unmount requests are sysplex-wide

- File system availability

- Recovery in place for maintaining file system availability. Dead System Recovery is provided to dynamically recover file systems owned by a system that has left the sysplex for any reason.

- Latches serialize file system operations and I/O to CDS

- Although latches are system-specific, they can lead to sysplex-wide hangs in a shared FS environment

- CDS serializes shared FS activity across the sysplex

In a shared FS configuration, it's important to understand that mounts and unmounts are sysplex-wide. When a file system is mounted on 1 system in a shared FS sysplex, it is "catchup" mounted on the remaining active systems in the shared FS sysplex. For HFS file systems mounted as RDWR (read/write), the actual physical I/O is done from 1 owning system. XCF is used to "function ship" I/O requests for a file system from the client systems to the owning system. zFS file systems mounted as RDWR can be configured to avoid z/OS UNIX function-shipping (sysplex-aware).





# OMVS Dataspaces

---

- z/OS UNIX uses many dataspaces; some common ones follow:
  - **SYSZBPX1 - Kernel data**
  - **SYSZBPX2 - File system data**
  - SYSZBPX3 - Pipes
  - SYSZBPXC - Common INET sockets
  - **SYSZBPXX - CTRACE buffers**
  - SYSZBPXU - AF\_UNIX sockets
  - BPXDMxxx - Memory Map
  - BPXDSxxx - Shared Memory
  - **BPXFSCDS - FileSys I/O buffer for BPXFSCDS record in OMVS CDS (Shared FS)**
  - HFSDSPxx - DFSMS HFS control block caching

The HFS dataspaces (HFSDSPxx) typically consume the most space in a dump, and they can usually be excluded from dumps for z/OS UNIX issues. For many types of problems, you can get away with only dumping dataspaces SYSZBPX1 (kernel data), SYSZBPX2 (file system data), and SYSZBPXX (CTRACE buffers), along with dataspace BPXFSCDS (OMVS CDS data) if running in a shared FS configuration.



# z/OS UNIX Serialization - Latches

- z/OS UNIX serialization is accomplished with GRS latches
  - Latches are system-specific, and each system has its own set of latches
- z/OS UNIX latch set control blocks reside in OMVS private storage
  - **SYS.BPX.A000.FSLIT.FILESYS.LSN**
    - File system latch set - serializes mount, unmount, and other file system operations
    - Latch#2 = MOUNT Latch, Latch#8...n = File System Latch
  - **SYS.BPX.A000.FSLIT.QUIESCE.LSN**
    - File system quiesce latch set – serializes quiesce operations
    - Latch#8...n = File System Quiesce Latch
  - **SYS.BPX.A000.FSLIT.FILESYS.LSN.01 (02,03,...,0n)**
    - File latch sets – serializes file operations
  - **SYS.BPX.A000.PRTB1.PPRA.LSN**
    - Process latch set - serialize process and thread operations
- Refer to [z/OS MVS Diagnosis: Reference, Understanding UNIX System Services latch contention](#)

It's important to understand that z/OS UNIX latch set control blocks reside in OMVS private storage. So, when looking at a dump, the OMVS address space must be dumped in order to determine whether there is contention with a z/OS UNIX latch.

Although latches are system-specific, they can lead to a sysplex-wide hang in a shared FS environment, if a latch on a particular system is holding up serialized shared FS activity from completing.



# Auxiliary Address Spaces

---

- **BPXOINIT**
  - Runs /etc/init (/etc/rc) during OMVS initialization
    - Initialization script
  - Handles MODIFY BPXOINIT console commands
  - Zombie cleanup
- **BPXAS Initiator**
  - Used when new address space is needed for fork processing
  - WLM-managed
  - Remain active for 30 minutes after last use
- **Colony Address Space**
  - Address space in which Physical File System can be initialized (e.g. zFS, NFS, TFS)

There are other address spaces, aside from the OMVS address space, that are part of the z/OS UNIX environment. The BPXOINIT address space is automatically created as part of OMVS initialization, and it runs the /etc/init OMVS initialization script. BPXOINIT also handles 'F BPXOINIT' console commands and cleans up zombie processes.

BPXAS initiators are WLM-managed initiators in which forked processes run. A BPXAS initiator is terminated after 30 minutes of inactivity. However, idle BPXAS initiators can be manually shutdown via 'F BPXOINIT,SHUTDOWN=FORKINIT'.

Some PFS's are initialized in separate address spaces called colony address spaces.

---



# Product Interaction

---

- LE C/C++ RTL to support POSIX API
- TSO/ISPF
  - OMVS shell and ISHELL
- RACF or OEM for security
- TCP/IP for network access
- WLM for fork initiator management (BPXAS)
- GRS for latch operations
- XCF for cross system requests (Shared FS sysplex)
- Various Physical File Systems (zFS, HFS, DFS, NFS, TFS, XPFS, TCP/IP)
- Major e-Business applications
  - Websphere, DB2, JAVA, CICS, IMS

---

z/OS UNIX interacts with many different product components, including PFS address spaces for file system operations, XCF for shared file system operations, and GRS for latch operations. The LE C/C++ Run-Time Library helps support the POSIX API.



# z/OS UNIX Syscalls

- z/OS UNIX API provided by set of syscalls
  - AMODE 31 – BPX1xxx or BPX2xxx
  - AMODE 64 – BPX4xxx
- Documented in [z/OS UNIX Programming: Assembler Callable Services Reference](#)
  - Function
  - Format
  - Parameters
    - Function specific parms
    - Return Value - (rv)
    - Return Code - (errno)
    - Reason Code - (errnojr)
  - Usage Notes

z/OS UNIX provides a set of Assembler Callable Services, also known as system calls (syscalls). Detailed information about the syscalls and parameters passed to the syscalls can be found in the [z/OS UNIX Programming: Assembler Callable Services Reference](#). Although parameters differ for each of the syscalls, the 3 R's (Return Value, Return Code, Reason Code) are returned for each syscall request.



# Syscall Return Information

---

- z/OS UNIX syscalls return at least three different values:
  - The **return value** (**rv**) is set to -1 (x'FFFFFFFF') if an error condition is found
  - The **return code** (**errno**) contains an integer value generally associated with a standard POSIX error
  - The **reason code** (**errnojr / errno2**) contains a four-byte value that provides specific information about the failure
- In general, messages or traces contain the corresponding return value, return code and reason code

Typically, the return value (rv) is set to -1 (x'FFFFFFFF') if an error occurs. If the return value is -1, then the return code (errno) and reason code (errnojr) provide more specific information about the failure. These “3 R’s” typically are externalized in messages or in traces (e.g. OMVS CTRACE, application trace).



# **z/OS UNIX**

## **Messages and Codes**

---



# z/OS UNIX Return/Reason Codes

---

- Refer to [z/OS UNIX Messages and Codes](#)
- z/OS UNIX Return Code (errno) generally corresponds to standard POSIX error, e.g. EAGAIN, EACCES, EBUSY
- z/OS UNIX Reason Code (errnojr /errno2)
  - A full word (**CCCCRRRR**)
  - **CCCC** is the reason code qualifier - component
  - **RRRR** is the reason code
  - If CCCC is between 0-20FF then RRRR is a z/OS UNIX reason code
    - Use **BPXMTEXT** from TSO, IPCS or Shell
- Reason codes from other components
  - See [z/OS UNIX Messages and Codes: Reason Codes \(Errnojrs\) Listed by Value](#)

**BPXMTEXT** is a very useful tool that can be invoked from TSO, the Shell, or under IPCS. For reason codes issued by z/OS UNIX (reason code qualifier in the range 0-20FF), BPXMTEXT provides the reason code name and issuing module, as well as a description of the reason code and what actions to take. Refer to the [z/OS UNIX Command Reference](#) for more details about the use of bpxmtext from the Shell.



---



# Reason Code Qualifiers for Other Components

---

- 5B00 - 5BFF: DFSMS HFS File System
- 6C00 - 6CFF: Distributed File Systems Client (DFSC)
- 6E00 - 6EFF: NFS Client File System
- 7100 - 71FF: VTAM Anynet Sockets
- 7300 - 73FF: z/OS Communications Server TCP/IP Stack
- 7880 - 78FF: z/OS Communications Server Resolver
- C000 - CFFF: Language Environment (C/C++ RTL)
- DF00 - DFFF: Distributed File Service File Exporter Exit Routine (IOEGLUE) and DFSKERN
- EF01 - EFFF: zFS File System

This slide shows common reason code qualifiers (first half of errnojr) and their respective components.

# Example Usage: BPXMTEXT

- BPXMTEXT can be used to quickly interpret an `errnojr` from z/OS UNIX, zFS, TCP/IP, and C/C++ RTL
  - TFS reason codes added at z/OS 2.2
- Shipped in `SYS1.SBPXEXEC` which must be in `SYSEXEC` to use
- USAGE: **BPXMTEXT *errnojr***
  - Use as either a TSO, Shell, or IPCS command
- **TSO BPXMTEXT 058800AA**

BPXFSUMT 12/05/14

JRFsParentFs: The file system has file systems mounted on it

Action: An unmount request can be honored only if there are no file systems mounted anywhere on the requested file system. Use the `F BPXOINIT, FILESYS=DISPLAY,ALL` command for a shared file system configuration or the `D OMVS,FILE` command for a non-shared file system configuration to determine which file systems are mounted on the requested file system. Unmount them before retrying this request. Also check the system log for message `BPXF271I`, which will identify the first mounted file system found.

In this example, a customer tried to unmount a file system with other file systems mounted on it. They received a z/OS UNIX reason code of `x'058800AA'`. Per BPXMTEXT output, the reason code name is 'JRFsParentFs' and indicates that the file system has file systems mounted on it.



# ETCINIT Exit Status Codes

---

- Failure reported to console by /usr/sbin/init (/etc/init) during OMVS initialization
  - **BPXI027I THE ETCINIT JOB ENDED IN ERROR, EXIT STATUS 0000xx00**
    - Refer to [z/OS UNIX Messages and Codes: Exit Status Codes for /usr/sbin/init](#)
  - If Exit Status of the form **000000xx**, xx is the signal number of the signal that ended the /usr/sbin/init process
    - Refer to [z/OS UNIX Programming: Assembler Callable Services Reference: Signal defaults](#)
    - Note: See also *Appendix A* for Signal Numbers
- Example:
  - BPXI027I THE ETCINIT JOB ENDED IN ERROR, EXIT STATUS [00000009](#)
    - /usr/sbin/init process ended by [signal number 09 \(09 = SIGKILL\)](#)

In this example, the ETCINIT job was ended abnormally via a SIGKILL.

---



# z/OS UNIX Abend Codes

---

- Refer to [z/OS MVS System Codes](#)
- Hexadecimal reason code describes the error
  - Can **NOT** use BPXMTEXT for z/OS UNIX abend reason codes
- **SEC6 (abendEC6)**
  - Reason Code: CCCC RRRR
  - RRRR values
    - For example: 0000FDxx ,0000FFxx
      - If xx is in the range of x'01' to x'7F', a signal was received
- **S422 (abend422)**
  - z/OS UNIX Reason Code: xxxx01zz
    - If zz is in the range of x'01' to x'7F', a signal was received

---

Unlike z/OS UNIX reason codes, z/OS UNIX abend codes cannot be deciphered with BPXMTEXT. The z/OS UNIX abend reason codes are documented in the [z/OS MVS System Codes](#) manual.

Refer to '[z/OS UNIX Programming: Assembler Callable Services Reference: Signal defaults](#)' for a list of signals and their default actions.

Note: See also **Appendix A** for a list of signals and signal numbers.

A signal is a mechanism by which a process may be notified of, or affected by, an asynchronous event occurring in the system. Delivery of a signal is accomplished via an IRB interrupt.

---



# Messages in z/OS UNIX Environment

---

- BPX - z/OS UNIX messages
- FSUM - z/OS UNIX Shell & Utilities messages
- FDBX - Debugger (DBX) messages
- FOM - Application Services messages
- EDC - LE C/C++ RTL messages
- CEE - LE messages
- IOE - DFS/zFS messages
- IGW - DFSMS/HFS messages
- EZx - TCP/IP messages

---

Some of the more commonly seen message prefixes in a z/OS UNIX environment are displayed on this slide and the next. z/OS UNIX messages are prefixed with BPX, and Shell messages are prefixed with FSUM.

---



# Messages in z/OS UNIX Environment (cont)

---

- GFSC/GFSA - NFS Client/Server messages
- ICH/IRR - RACF messages
- IMW - WebSphere messages
  
- **LOOKAT** Message Tool (available for messages/ABENDs through z/OS 1.13)
  - <http://www-03.ibm.com/systems/z/os/zos/bkserv/lookat/index.html>



# **z/OS UNIX**

## **Console Commands**



## D OMVS,A=ALL

- Displays all currently active z/OS UNIX processes on the system
  - BPXO047I – USER, JOBNAME, ASID, PID, PPID, STATE, CMD
- Use to find userid associated with jobname for OMVS CTRACE filtering
- Also useful for identifying zombie buildup
  - Zombie processes have a state of **1Z**
  - If zombies have parent with PID=1 (PPID=1), then it is responsibility of BPXOINIT to clean them up
    - Issue **D J,BPXOINIT** to verify that BPXOINIT is swapped in and running
  - If parent of zombies does not have PID=1, then it is responsibility of parent to cleanup zombies
    - Did parent process issue a **waitpid()** to wait for child processes to end?

The 'D OMVS,A=ALL' command displays all active z/OS UNIX processes on the system.

This output is useful, for example, to identify a zombie buildup. When a dubbed process terminates, process blocks are kept around until status can be reported to its parent. A process in this state is called a "zombie".



---



## D OMVS,F

---

- Displays all mounted file systems from this system's perspective
  - BPXO045I - NAME, PATH, MODE, TYPENAME, STATUS, LATCHES, QSYSTEM, QPID, QJOBNAME, OWNER, AUTOMOVE, CLIENT
- Use to check for quiesced file systems
- Also useful in shared FS environment to compare each system's local view of mounted file systems

---

The OWNER, AUTOMOVE, and CLIENT fields are only displayed in a shared FS environment.

---



## D OMVS,L and D OMVS,L,PID=

- **'D OMVS,L'** displays information about system limits from BPXPRM<sub>xx</sub>, including their high-water marks and current system usage
  - BPXO51I
- **'D OMVS,L,PID='** displays information about process limits, including high-water marks and current usage for an individual process
  - BPXO51I
- Use to check whether system or process limit is being reached

The 'D OMVS,L' display is useful for checking whether a system limit (e.g. MAXPROCSYS) has been reached, whereas the 'D OMVS,L,PID=' display is useful for checking with a process limit (e.g. MAXFILEPROC) has been reached.

---



## D GRS,C

---

- Check for any ENQ or latch contention on a system
  - ISG343I
- Use to check for any z/OS UNIX latch contention when diagnosing a hang in a z/OS UNIX environment
  - If hang involves MOUNT Latch, File System Latch, or File Latch, the '**D OMVS,W**' display can be used to get more details

The 'D GRS,C' command is especially useful when diagnosing a hang in a z/OS UNIX environment. It will show any ENQ or z/OS UNIX latch contention on a given system.



## D OMVS,W

---

- Displays information about threads waiting in z/OS UNIX environment
  - [BPXO063I](#)
- Following areas of delay are checked:
  - MOUNT Latch Activity
  - File System Latch Activity (only if contention)
  - File Latch Activity (only if contention)
  - Outstanding Cross-System Messages
    - Received messages
    - Sent messages
  - Other Waiting Threads
- Particularly useful when diagnosing a hang in a z/OS UNIX environment involving contention with the MOUNT Latch, File System Latch, or File Latch

The 'D OMVS,W' output is extremely valuable when trying to determine the reason for hangs or delays in a z/OS UNIX environment. When 'D GRS,C' shows contention with the Mount Latch, a File System Latch, or a File Latch, the 'D OMVS,W' output will provide additional information about the file system and operation involved. It also shows any outstanding cross-system messages (received and sent), and it provides a table of other waiting threads.



# F BPXOINIT Shared FS Diagnostics

- **F BPXOINIT,FILESYS=D,GLOBAL**
  - **BPXF242I** - Displays current sysplex state, including active systems (system name, LFS version level), and active serialization categories
    - SYSTEM PERFORMING INITIALIZATION
    - SYSTEM PERFORMING MOVE
    - SYSTEM PERFORMING QUIESCE
    - SYSTEMS PERFORMING UNMOUNT
    - SYSTEM PERFORMING REMOUNT
    - SYSTEMS PERFORMING MOUNT RESYNC
    - SYSTEMS PERFORMING LOCAL FILE SYSTEM RECOVERY
    - SYSTEMS PERFORMING FILE SYSTEM TAKEOVER RECOVERY
    - SYSTEMS RECOVERING UNOWNED FILE SYSTEMS
    - SYSTEMS PERFORMING REPAIR UNMOUNT
- Issue from system with highest LFS version level and no MOUNT Latch contention

LFS version level (LFS software version)

Used to determine if systems are compatible with other systems in the shared FS plex, and whether new functions can be enabled.

The GLOBAL display can be used to identify the z/OS release level of all systems currently active in the shared FS sysplex. It is particularly useful in a shared FS sysplex hang situation to identify whether any serialized function(s) are occurring, for how long, and on which system(s).



# F BPXOINIT Shared FS Diagnostics (cont)

## F BPXOINIT,FILESYS=D,ALL

- BPXF035I - Displays all file systems in the sysplex (CDS) shared FS hierarchy

## F BPXOINIT,FILESYS=D,FILESYSTEM=file\_system\_name

- BPXF035I - Displays information about file system from sysplex (CDS) perspective

## F BPXOINIT,FILESYS=D,EXCEPTION

- BPXF035I - Displays file systems in exception state, including differences between local and sysplex view
- Exception states include: Mount in progress, Unmount in progress, Quiesce in progress, Quiesced, Unowned , In recovery, Unusable

## F BPXOINIT,FILESYS=DUMP

- BPXF043I - Capture dump of OMVS and dataspace, including CDS sub-records
  - Should not be issued on system with MOUNT Latch contention

This slide lists some of the other 'F BPXOINIT,FILESYS=' commands and their uses.

The 'F BPXOINIT,FILESYS=D,ALL' command displays all file systems mounted in the shared FS hierarchy and should match what 'D OMVS,F' shows from any given system. The EXCEPTION command will show any discrepancies between a system's local view and the view from the OMVS CDS.



# F BPXOINIT Shared FS Diagnostics (cont)

- **F BPXOINIT,FILESYS=FIX**

- Note: Use with CAUTION and issue from system with highest LFS version, if possible
- Captures dump of OMVS and dataspaces, including sub-records in the CDS, prior to diagnosis and repair
- Performs automatic file system and CDS diagnosis and repair
- Typically used for display purposes to isolate problem system
- FIX processing has both synchronous processing from the system that the FIX was issued, and also asynchronous processing that occurs on all other systems in the shared FS sysplex
- FIX issues messages pointing to specific system(s) causing delay
  - Need to check the hardcopy system logs of all systems in the shared FS sysplex for any messages issued in response to FIX

The FIX command is most often used for display purposes to complement the output from the GLOBAL command. FIX processing runs on all systems in the shared FS sysplex, and it's necessary to check the hardcopy logs of all systems in the shared FS sysplex for any messages in response.



## F BPXOINIT Shared FS Diagnostics (cont)

- **F BPXOINIT,FILESYS=FIX** messages
  - Message **BPXF049I** for each file system that is delayed during unmount, quiesce, or remount processing. The message also lists the systems that are causing the delay.
  - Message **BPXF042I** for each system that has contention for the file system MOUNT latch. Contention for the MOUNT latch delays high-level functions, such as mount and unmount processing.
  - Message **BPXF057I** for each file system that has latch contention. The message identifies the file system and the system where the latch contention is occurring.
  - Hardcopy message **BPXF048I** for each correction made to the file system global data structures (in the BPXMCDS couple data set).

If the GLOBAL output indicates that a system is delayed during unmount, quiesce, or remount serialized activity, the FIX processing will determine whether the systems performing those serialized functions are delayed due to other system(s). If so, message BPXF049I identifies the systems that have not yet performed the specified operation locally.



---



# **z/OS UNIX PD Documentation**

# Console Dump of OMVS and ZFS

- Console Dump Example:

```
DUMP COMM=(description or dump name)
R xx,JOBNAME=(OMVS,ZFS,other_process_jobnames),CONT
R xx,SDATA=(PSA,ALLNUC,LPA,CSA,SQA,LSQA,RGN,TRT,GRSQ,SUM),CONT
R xx,DSPNAME=('OMVS'.B*,'OMVS'.S*),END
```

- Instead of (or in conjunction with) JOBNAME, ASID=(omvs\_asid,zfs\_asid,other\_process\_asids) may be used
- `DSPNAME=('OMVS'.B*,'OMVS'.S*)` will not capture HFS dataspaces that typically take up a lot of space in the dump dataset
- May need to use wildcard '\*' for jobname of forked/spawned address spaces

For fork and spawn requests that do not involve changing user IDs, the jobname of the child is set to the base jobname with a number from 1 to 9 appended at the end. For example, if you logon to TSO, you will have a jobname that is the same as your user ID (for example, JSHEBEY). The first fork or spawn creates an address space with JSHEBEY1. In this case, the base jobname is JSHEBEY and all children inherit the same base jobname. Continuing this example, if address space JSHEBEY1 does a fork or spawn, the new child address space will have a jobname of JSHEBEY1 to JSHEBEY9. It is possible to have multiple address spaces with the same jobname running concurrently. This is why the wildcard '\*' is often needed for dumping the jobname of a forked/spawned address space.

# Console Dump of OMVS and ZFS on All Systems in Shared FS Sysplex

- Example of getting console dumps on all systems in shared FS sysplex:

```
DUMP COMM=(description or dump name)
JOBNAME=(OMVS,ZFS),DSPNAME=('OMVS'.B*,'OMVS'.S*),
SDATA=(PSA,ALLNUC,LPA,CSA,SQA,LSQA,RGN,TRT,GRSQ,SUM),
REMOTE=(SYSLIST=('OMVS','ZFS'),SDATA,DSPNAME)
```

- Issue command on **only one** system in shared FS sysplex, and dump of OMVS and its dataspace, along with ZFS, will be taken on all systems in shared FS sysplex
- SYS1.PARMLIB(IEADMCxx) - dump Parmlib member
  - **DUMP PARMLIB=xx**

This console dump command only needs to be issued on 1 system in a shared FS sysplex configuration. Through use of the REMOTE parameter, dumps of OMVS and its dataspace are collected on each system in the shared FS sysplex.

# SLIP to Dump OMVS and ZFS on All Systems in Shared FS Sysplex

- The following SLIP trap makes use of the REMOTE parameter to dump OMVS and ZFS on all systems in the shared FS sysplex.
- In this example, the SLIP trap will dump OMVS and its dataspace, along with ZFS, on all systems in the shared FS sysplex on the occurrence of an abend30D:

```
SLIP SET,COMP=30D,ACTION=SVCD,ID=S30D,  
DSPNAME=('OMVS'.B*,'OMVS'.S*),JL=(OMVS,ZFS),  
SDATA=(PSA,ALLNUC,LPA,CSA,SQA,LSQA,RGN,TRT,GRSQ,SUM),  
REMOTE=(SYSLIST=(SYSBPX.*),SDATA,JOBLIST,DSPNAME,A=SVCD),END
```

- Note the difference in syntax with the REMOTE parameter as compared to the console dump

Note the difference in syntax for the REMOTE parameter on a SLIP versus a console DUMP command.

## z/OS UNIX Reason Code SLIP: Pre z/OS 2.2

- Generic SLIP trap can be used to get dump of specific z/OS UNIX reason code (including PFS reason codes):

```
SLIP SET, IF, A=SYNCSVCD, RANGE= (10?+8C?+F0?+1F4?) ,  
DATA= (13R??+1B0, EQ, xxxxxxxx) , DSPNAME= ('OMVS' .B*, 'OMVS' .S*) ,  
SDATA= (ALLNUC, PSA, CSA, LPA, TRT, SQA, RGN, SUM) ,  
J=jobname, JL=OMVS, END
```

- where '**xxxxxxxx**' is the 8 digit (4 byte) reason code that is to be trapped and 'jobname' is the optional jobname associated with the error
- use wildcard '\*' for jobname of forked/spawned address spaces
- IEASLPxx in SYS1.PARMLIB can be used to store the slip
  - **SET SLIP=xx**
- Documented in [z/OS UNIX Messages and Codes](#)

This SLIP trap can be used to get a dump for a specific reason code issued by z/OS UNIX (reason code qualifier in the range 0-20FF) or from a PFS reason code (e.g. zFS, HFS, NFS, TCPIP).

## z/OS UNIX Reason Code SLIP: z/OS 2.2

- Generic SLIP trap can be used to get dump of specific z/OS UNIX reason code (including PFS reason codes):

```
SLIP SET, IF, A=SYNCSVCD, RANGE= (10?+8C?+F0?+1F4?) ,  
DATA= (13R!!+1B0, EQ, xxxxxxxx) , DSPNAME= ('OMVS' .B*, 'OMVS' .S*) ,  
SDATA= (ALLNUC, PSA, CSA, LPA, TRT, SQA, RGN, SUM) ,  
J=jobname, JL=OMVS, END
```

- where 'xxxxxxx' is the 8 digit (4 byte) reason code that is to be trapped and 'jobname' is the optional jobname associated with the error
- use wildcard '\*' for jobname of forked/spawned address spaces
- IEASLPxx in SYS1.PARMLIB can be used to store the slip
  - **SET SLIP=xx**
- Documented in [z/OS UNIX Messages and Codes](#)

The z/OS UNIX reason code SLIP at z/OS 2.2 is identical to that of z/OS 1.13 and z/OS 2.1, with the exception of addressing Register 13 (highlighted above in red) in the DATA= parameter. With z/OS 2.2, the DATA operand must change to accommodate a 64-bit Register 13, since we can be running with stacks above.



# Collecting OMVS CTRACE Records

- The OMVS CTRACE is valuable for many types of problems
  - History of syscalls leading up to the dump (e.g. SLIP event, ABEND)
- Turn on OMVS CTRACE with console command
  - **TRACE CT,64M,COMP=SYSOMVS**
  - **R xx,OPTIONS=(ALL),END**
    - **JOBNAME** can be specified (refers to “userid” associated with job or user)
      - For OMVS kernel trace entries, specify JOBNAME of **OMVS**
- Verify OMVS CTRACE status
  - **D TRACE,COMP=SYSOMVS**
- Turn off CTRACE after dump collected
  - **TRACE CT,OFF,COMP=SYSOMVS**
- OMVS CTRACE records can be sent to External Writer to cover more time
  - See *Appendix B*

OMVS CTRACE data can be filtered by OPTIONS or by JOBNAME. When filtering by JOBNAME, it is important to note that this filtering is based on the userid of the job, not its jobname. The userid associated with a job can be obtained from ‘D OMVS,A=ALL’ output. However, be aware that the OMVS Kernel is traced with a JOBNAME of OMVS.

Refer to [z/OS MVS Diagnosis: Tools and Service Aids](#) for a description of what OPTIONS can be used for filtering SYSOMVS CTRACE data.

# OMVS CTRACE at IPLTime

- The **CTIBPX00** parmlib member is used to start the OMVS CTRACE at IPL time.
  - Uncomment the 'ON' statement, the 'ALL' options statement, and set the BUFSIZE as required with the max being **64M**
    - CTIBPX00 is the default member for OMVS CTRACE startup. The BPXPRMxx member CTRACE statement may have been used to designate a different OMVS CTRACE parmlib member, so you will need to check in BPXPRMxx before assuming the default.
- CTRACE usage is documented in [z/OS MVS Diagnosis: Tools and Service Aids](#)

```
/* ----- */
/* DEFAULT CTIBPX00 MEMBER          */
/* ===== */
TRACEOPTS
/* ----- */
/* ON OR OFF: PICK 1                */
/* ----- */
    ON
/*    OFF                            */
/* ----- */
/* ASID: 1 TO 16, 2-HEXBYTE VALUES */
/* ----- */
/*    ASID(0042,0043,0044)          */
/* ----- */
/* BUFSIZE: A VALUE IN RANGE 16K TO 4M */
/* ----- */
    BUFSIZE(64M)
/* ----- */
/* OPTIONS: NAMES OF FUNCTIONS TO BE TRACED, OR "ALL", OR "NONE" */
/* ----- */
    OPTIONS(
        'ALL '
    )
```





# **z/OS UNIX PD Procedures**



---

# Abends and Failing Reason Codes

---

- Refer to INFO APAR **II08038** for general doc collection procedures
- **Abends (i.e. abendEC6)**
  - Normally generate SVC dump
    - Do not normally get dumps for abendEC6 rsn0000FFxx
  - If no SVC dump generated, may need to SLIP on abend
  - May need OMVS CTRACE
  - May need SYSLOG and EREP data
- **Failing Reason Code**
  - Often need SLIP dump for reason code
  - May need OMVS CTRACE
  - May need SYSLOG and EREP data

INFO APAR **II08038** contains general z/OS UNIX doc collection procedures.



# Hang in Non-Shared FS Environment

- Issue '**D GRS,C**'
- Issue '**D OMVS,W**' if latch contention (MOUNT/File System/File Latch)
  - Look for file system involved and associated PFS
  - [z/OS MVS Diagnosis: Reference, Procedure: Diagnosing and resolving latch contention](#)
- Need console dump of OMVS and its dataspace, along with:
  - Any non-OMVS latch holders (if z/OS UNIX latch contention)
  - Any hung jobs and/or users
  - Potentially any related PFS address spaces (e.g. ZFS, NFS, TCPIP) on failing system
- May need SYSLOG and EREP data
  - SYSLOG would contain output from above display commands

The 'D GRS,C' and 'D OMVS,W' displays can be used together when diagnosing a hang in a non-shared FS environment.

# Hang in Shared FS Environment

- **F BPXOINIT,FILESYS=DISPLAY,GLOBAL**
  - Identifies active systems (system with highest level LFS code), and shows if system(s) performing serialized activity
    - If systems performing **UNMOUNT, REMOUNT, or QUIESCE**, then '**F BPXOINIT,FILESYS=FIX**' can be issued to further isolate the problem system
      - Check hardcopy log of system(s) performing serialized activity for message **BPXF049I**
- **RO \*ALL,D GRS,C**
  - Look for contention on latch set **SYS.BPX.A000.FSLIT.FILESYS.LSN**, especially with Latch #2 (MOUNT Latch), and latch set **SYS.BPX.AP00.PRTB1.PPRA.LSN**
- **RO \*ALL,D OMVS,W**
  - If MOUNT, File System, or File Latch contention, output shows information about holder of latch, including file system involved (if any) and operation; look at how long latch held
  - Look for any long outstanding cross-system message requests

In a shared FS environment, the 'D GRS,C' and 'D OMVS,W' commands should be routed to all systems. The GLOBAL display also shows whether any systems(s) are stuck performing serialized function(s) that could be hanging up the shared FS sysplex.



## Hang in Shared FS Environment (cont)

- [z/OS MVS Diagnosis: Reference](#) has detailed information about shared FS diagnosis and recovery for various problem scenarios
  - Diagnostic Procedures for Shared File System
  - Procedure: Diagnosing and resolving latch contention
- Capture EREP data and SYSLOGs for every system in the shared FS sysplex – a merged OPERLOG is preferable
  - The SYSLOGs should contain all of the above information
- Capture sysplex-wide dumps of OMVS and its dataspace
  - Dump any hung address spaces involved in problem
  - On systems with latch contention, be sure to dump any non-OMVS latch holders separately
  - Also dump any PFS address spaces that may be related to the problem (e.g. zFS, NFS, TCPIP)

Sysplex-wide dumps of OMVS and its dataspace, along with any other related PFS's or address spaces, are often needed when debugging a hang in a shared FS environment.

The 'D OMVS,W' output can help pinpoint more specific systems involved, but if in doubt, dumps should be collected from all systems.

---



# OMVS Initialization Hang

---

- Review SYSLOG and look for message BPXP006E
- **BPXP006E OMVS IS:**
  - INITIALIZING THE FILE SYSTEM
  - CREATING THE BPXOINIT ADDRESS SPACE
  - PROCESSING IN BPXOINIT
  - STARTING THE INITIALIZATION PROCESS
  - RUNNING THE INITIALIZATION PROCESS
  - WAITING FOR SECURITY PRODUCT INITIALIZATION
  - WAITING FOR CATALOG ADDRESS SPACE INITIALIZATION
  - WAITING FOR JOB ENTRY SUBSYSTEM INITIALIZATION
  - OMVS IS UNABLE TO CREATE THE BPXOINIT ADDRESS SPACE

When OMVS initialization hangs, message BPXP006E may be issued identifying which area of OMVS initialization is hung.



## OMVS Initialization Hang (cont)

- If OMVS is RUNNING THE INITIALIZATION PROCESS, review `/etc/log` to see output from `/etc/rc` and last command that successfully ran
- Also review SYSLOG for message BPXP007E
  - **BPXP007E STARTING PHYSICAL FILE SYSTEM *pfsname* IN ADDRESS SPACE *spacename***
- In shared FS sysplex, may instead see message BPXF076I
  - **BPXF076I FILE SYSTEM INITIALIZATION IS DELAYED DUE TO CONFLICTING ACTIVITY ON ANOTHER SYSTEM.**
    - Check '**F BPXOINIT,FILESYS=D,GLOBAL**' output for shared FS serialized activity on another system
- A console dump of OMVS and its dataspace may be needed (sysplex-wide dumps in shared FS sysplex), along with any address spaces involved with the hang
- SYSLOG and EREP should be sent for history of messages/abends during IPL

If a PFS address space gets hung, holding up OMVS initialization, message BPXP007E will be issued. In a shared FS sysplex, it's possible for a system to be hung in initialization due to other serialized activity in the shared FS sysplex (message BPXF076I).

---



# **z/OS UNIX**

## **Problem Examples**



# Problem #1: Terminating Signal

- Message BPXP023I (in SYSLOG) indicates source of signal and receiver of signal
  - Thread ID in hex
  - Process ID in decimal.
    - PID 67108975 = x'0400006F' (for OMVSDATA in dump)
    - PID 16777241 = x'01000019' (for OMVSDATA in dump)

```
BPXP023I THREAD 352E800000000000, IN PROCESS 67108975,  
WAS TERMINATED BY SIGNAL SIGKILL, SENT FROM THREAD  
0000000000000000, IN PROCESS 16777241, UID 0, IN JOB KILLER.
```

```
IEF450I G113821 STEP1 - ABEND=SEC6 U0000 REASON=0000FF09
```

- Abend EC6 REASON=0000FF09
  - FF09 = SIGKILL

In this example, the thread with ID x'352E800000000000' in process ID 67108975 was terminated via a SIGKILL signal from process ID 16777241 with jobname KILLER. This resulted in the target process terminating abnormally with an ABENDEC6 RSNFF09.

## Problem #2: OMVS Initialization Error

- Customer received following error during IPL of SY1:
  - **BPXI027ITHE ETCINIT JOB ENDED IN ERROR, EXIT STATUS 00000900**
  - Per z/OS UNIX Messages and Codes: Exit Status Codes for /usr/sbin/init

### 09 No Stdout

/usr/sbin/init could not open /etc/log for standard output.  
Contact your system programmer. The system continues.

Note: /usr/sbin/init attempts to erase and re-create /etc/log  
each time it is run.

- Possible Causes
  - Full file system, improper authorization, or read-only /etc file system

The ETCINIT job failed during OMVS initialization with an exit status code of 00000900. Looking up the meaning of exit status code '09', it indicates that /etc/log could not be opened for output.

# Problem #2: OMVS Initialization Error

## D OMVS,F

- Use D OMVS,F to display /etc file system for SY1

```
SY1  D OMVS,F
OMVS      0011 ACTIVE                OMVS=(PD, BB, LA)
TYPENAME  DEVICE  -----STATUS-----  MODE  MOUNTED  LATCHES
ZFS        1311 ACTIVE                READ  07/13/2016  L=1206
          NAME=ZOS22.SY1.ETC.ZFS      16.28.16  Q=1206
          PATH=/SY1/etc
          MOUNT PARM=FSFULL(90,1)
          OWNER=SY1      AUTOMOVE=U CLIENT=Y
```

- Note that SY1 /etc file system 'ZOS22.SY1.ETC.ZFS' is mounted as read-only

We see from 'D OMVS,F' output that the /etc file system is mounted as READ (read-only), so /etc/log (file in /etc file system) cannot be opened for output (write).

---



## Problem #3: System Limit Exceeded

- Attempts to start new OMVS processes failed as follows:
  - **BPXP012I FORK SYSCALL TERMINATED DURING CHILD PROCESSING WITH RETURN CODE 000009C, REASON CODE 0B0F0028.**
- RC x'9C' - EMVSINITIAL - Process Initialization Error
- **TSO BPXMTEXT 0B0F0028**

```
BPXPRTB1 09/07/11
```

```
JRMaxProc: The maximum number of processes was exceeded
```

```
Action: Retry after some processes have ended, or change  
the maximum number of processes allowed.
```

In this example, a fork failed with rsn0B0F0028 JRMaxProc, which is indicative of the MAXPROCSYS limit value being reached.

# Problem #3: System Limit Exceeded

## D OMVS,L

MAXPROCSYS limit value can be raised for the current IPL via the SETOMVS command

- SETOMVS  
MAXPROCSYS=nnnnn

```

D OMVS,L
BPX0051I 06.22.07 DISPLAY OMVS 171
OMVS      000D ACTIVE          OMVS=(6E)
SYSTEM WIDE LIMITS:          LIMMSG=NONE

```

	CURRENT USAGE	HIGHWATER USAGE	SYSTEM LIMIT
MAXPROCSYS	256	256	256
MAXUIDS	0	0	200
MAXPTYS	0	0	256
MAXMAPAREA	0	0	256
MAXSHAREPAGES	0	10	4096
IPCMSGNIDS	0	0	500
IPCSEMNIDS	0	0	500
IPCSEMNIDS	0	0	500
IPCSEHMPAGES	0	0	262144
IPCMSGQBYTES	---	0	2147483647
IPCMSGQNUM	---	0	10000
IPCSEHMPAGES	---	0	25600
SHRLIBRGNSIZE	0	0	67108864
SHRLIBMAXPAGES	0	0	4096
MAXUSERMOUNTSYS	15	20	100
MAXUSERMOUNTUSER	7	8	10
MAXPIPES	23	521	15360

The 'D OMVS,L' display can be used to confirm that we have reached the MAXPROCSYS limit value. We can temporarily (for the current IPL) raise the MAXPROCSYS limit value via the SETOMVS command to provide immediate relief.

# Problem #3: System Limit Exceeded

## D OMVS,A=ALL

- One reason for exceeding MAXPROCSYS is due to a buildup of zombies
- **D OMVS,A=ALL**

```
D OMVS,A=ALL
BPXO040I 17.25.48 DISPLAY OMVS 202
OMVS      000D ACTIVE          OMVS=(6F)
USER      JOBNAME  ASID      PID      PPID STATE   START    CT_SECS
WASPISRV IMWEBJBP 0021      196637      1 MR---- 10.08.13    .629
  LATCHWAITPID=      0 CMD=IMWHTPD
WSPUBLIC   0000 0050528627    196637 1Z---- 14.02.48    .001
WSPUBLIC   0000 0084083060    196637 1Z---- 14.07.10    .001
WSPUBLIC   0000 0016974201    196637 1Z---- 11.00.57    .001
WSPUBLIC   0000 0033751418    196637 1Z---- 14.07.00    .001
WSPUBLIC   0000 0000196987    196637 1Z---- 11.35.43    .001
...
```

- Note that there is a buildup of zombies (STATE=1Z), all with a parent PID of 196637
- This should be pursued by Webserver support

If there is a zombie buildup (state of 1Z), this can cause the process slots to be filled up, and MAXPROCSYS reached. The most probable reason for a zombie buildup is that a parent process does not issue a waitpid() – BPX1WAT – on its child processes. In such a case, the parent PID of the zombies would be equal to the ID of the process that created them. In this example, they have a parent PID of 196637, which is associated with an HTTP server.

---



## Problem #4: Shared FS Hang

---

- Customer was running with 7 z/OS 2.1 systems in a shared FS sysplex. At the time of the problem, 6 systems were active (TEST was down).
- Customer attempted to bring TEST back into the shared FS plex, but OMVS initialization hung, and they got message BPXF076I:
  - **BPXF076I FILE SYSTEM INITIALIZATION IS DELAYED DUE TO CONFLICTING ACTIVITY ON ANOTHER SYSTEM.**

This is an example where a system (TEST) hung in OMVS initialization due to serialized shared FS activity on another system in the sysplex.

---



## Problem #4: Shared FS Hang (cont)

- The '**F BPXOINIT,FILESYS=D,GLOBAL**' command showed the following:

```
SYSTEMS PERFORMING LOCAL FILE SYSTEM RECOVERY
(Since 2016/07/01 13.52.02)
DEV
SYSTEMS PERFORMING FILE SYSTEM TAKEOVER RECOVERY
(Since 2016/07/01 13.52.02)
DEV
```

- Note that Dead System Recovery processing on system DEV was ongoing.

The GLOBAL display shows that Dead System Recovery (FILE SYSTEM RECOVERY) is currently processing on another system (DEV).



## Problem #4: Shared FS Hang (cont)

- TEST got message BPXF076I during its IPL into the shared FS sysplex because it couldn't complete its Initialization processing until the Dead System Recovery processing on DEV completed, since they are conflicting activity.
- The '**D GRS,C**' display (routed to all systems in shared FS plex) showed contention with the MOUNT Latch on system DEV.

```
D GRS,LATCH,C
ISG343I 15.53.07 GRS STATUS 034
LATCH SET NAME:  SYS.BPX.A000.FSLIT.FILESYS.LSN
CREATOR JOBNAME: OMVS      CREATOR ASID: 0011
LATCH NUMBER:  2
  REQUESTOR  ASID  EXC/SHR   OWN/WAIT  WORKUNIT  TCB  ELAPSED TIME
  OMVS       0011 EXCLUSIVE OWN       008F8A68  Y   04:17:38.662
  OMVS       0011 EXCLUSIVE WAIT      008D9580  Y   02:00:02.898
```

- Note that OMVS TCB(x'8F8A68') holds the MOUNT Latch.

There is MOUNT Latch contention on the system (DEV) that is stuck performing Dead System Recovery.

# Problem #4: Shared FS Hang (cont)

The 'D OMVS,W' output from DEV tied the MOUNT Latch holder to the Dead System Recovery processing, and it showed the latch holder waiting on a PPRP (e.g. PPRA) latch.

```
RO DEV,D OMVS,W
BPX0063I 15.53.05 DISPLAY OMVS 419
OMVS      0011 ACTIVE          OMVS=(00,01,0D,DC)
MOUNT LATCH ACTIVITY:
  USER  ASID   TCB          REASON                AGE
-----
HOLDER:
  OMVS   0011  008F8A68    MemberGone Rcvry      02.00.00
  TIME: 2016/07/01 13.52.05
  IS DOING: PPRP Latch Wait
  FILE SYSTEM: SYSOS.ZFS.ROOT

OUTSTANDING CROSS SYSTEM MESSAGES:
RECEIVED SYSPLEX MESSAGES:
  ON TCB  FROM  FROM  FROM
  ASID   TCB  FCODE MEMBER REQID/SEQ MSG TYPE  AGE
-----
  008F8A68 0000 00000000 *0000          01000000 MemberGone 02.01.02
  TIME: 2016/07/01 13.52.05          00000000
  IS DOING: PPRP Latch Wait
  FILE SYSTEM: SYSOS.ZFS.ROOT
```

The 'D OMVS,W' output from DEV confirms that the MOUNT Latch contention and the Dead System Recovery hang are related. The MOUNT Latch holder is waiting on a PPRP latch.

## Problem #4: Shared FS Hang (cont)

- The 'D GRS,C' display on DEV shows the MOUNT Latch holder waiting for PPRA Latch#176, which is held by TCB(x'8C0E88') in ASID(x'8C') WSJS1:

```
D GRS,C
ISG343I 15.53.07 GRS STATUS

LATCH SET NAME:  SYS.BPX.AP00.PRTB1.PPRA.LSN
CREATOR JOBNAME: OMVS          CREATOR ASID: 0011
LATCH NUMBER:  176
  REQUESTOR  ASID  EXC/SHR  OWN/WAIT  WORKUNIT  TCB  ELAPSED TIME
  WSJS1      008C  EXCLUSIVE  OWN       008C0E88  Y   04:45:05.324
  OMVS       0011  EXCLUSIVE  WAIT      008F8A68  Y   04:17:01.557
```

- In addition to dumping OMVS and its dataspace, job WSJS1 should be included in the dump.
- To potentially recover the shared FS hang in sysplex, job WSJS1 may be canceled.

The 'D GRS,C' display shows the MOUNT Latch holder waiting for PPRA Latch#176, which is held by TCB(x'8C0E88') in ASID(x'8C') WSJS1.

It may be possible to cancel job WSJS1 to free up the shared FS sysplex.

---



## Problem #5: Shared FS Hang

---

- Customer was running with 24 z/OS 2.1 systems in a shared FS sysplex. At the time of the problem, 22 systems were active (SYS4 and SYS2 were down).
- Customer attempted to bring SYS2 back into the shared FS plex, but OMVS initialization hung, and they got message BPXF076I:
  - **BPXF076I FILE SYSTEM INITIALIZATION IS DELAYED DUE TO CONFLICTING ACTIVITY ON ANOTHER SYSTEM.**

This is another example where a system (SYS2) hung in OMVS initialization due to serialized shared FS activity on another system in the sysplex.

---



## Problem #5: Shared FS Hang (cont)

- The 'F BPXOINIT,FILESYS=D,GLOBAL' command showed the following:

```
SYSTEMS PERFORMING LOCAL FILE SYSTEM RECOVERY
(Since 2016/06/09 01.00.00)
  SYS3
SYSTEMS PERFORMING FILE SYSTEM TAKEOVER RECOVERY
(Since 2016/06/09 01.00.00)
  SYS3
SYSTEMS PERFORMING UNMOUNT
(Since 2016/06/08 14.30.00)
  SYS1
```

- Note that UNMOUNT processing on SYS1 is the earliest shared FS activity, hung up since the day before

The GLOBAL display shows that Dead System Recovery (FILE SYSTEM RECOVERY) is currently processing on another system (SYS3), as well as UNMOUNT processing (on SYS1) from the day before!

---



## Problem #5: Shared FS Hang (cont)

- SYS2 got message BPXF076I during its IPL into the shared FS sysplex because it couldn't complete its Initialization processing until both the Unmount processing on SYS1 and the Dead System Recovery processing on SYS3 completed, since they are conflicting activity.
- The '**D GRS,C**' display (routed to all systems in the shared FS plex) showed contention with the MOUNT Latch on systems SYS3, SYSA, and SYS6.
- In an attempt to further isolate the problem and determine why the Unmount processing on SYS1 was hung up, the customer issued '**F BPXOINIT,FILESYS=FIX**'

---

The FIX command can be used to determine whether the UNMOUNT processing on SYS1 is delayed due to another system.

## Problem #5: Shared FS Hang (cont)

- The following message was issued on SYS1 as part of FIX processing:

```
BPXF049I UNMOUNT PROCESSING FOR FILE SYSTEM  
POSIX.ETC.HFS REQUIRES RESPONSES FROM THE  
FOLLOWING SYSTEMS: SYS3
```

- The following message was issued on SYS3 as part of FIX processing:

```
BPXF042I POSSIBLE CONTENTION FOR THE FILE SYSTEM  
MOUNT LATCH EXISTS ON SYSTEM SYS3,  
LATCH NUMBER 2
```

- Taking into account the GLOBAL and FIX output, SYS3 is the problem system – IPL may be required

In this case, the FIX processing reports (message BPXF049I) that the UNMOUNT processing on SYS1 is waiting for a response from SYS3, and we know that SYS3 has MOUNT Latch contention. Thus, the problem system is SYS3, which may need to be IPL'd to recover.

---



# **z/OS UNIX**

## **Reference Information**



---



# Programming References

---

- [C/C++ Run-time Library Reference](#)
- [z/OS Using REXX and z/OS UNIX System Services](#)
- [z/OS UNIX Programming: Assembler Callable Services Reference](#)
  - BPX1xxx, BPX2xxx, BPX4xxx
- [z/OS UNIX Filesystem Interface Reference](#)
  - Virtual File System operations (vfs\_xxx and v\_xxx)
  - Vnode operations (vn\_xxx)
  - Operating System Interface (OSI)

These references are of particular use to programmers in a z/OS UNIX environment.

---



## Other z/OS UNIX References

---

- Manuals
  - [z/OS UNIX Command Reference](#)
  - [z/OS UNIX Messages and Codes](#)
  - [z/OS UNIX Planning](#)
  - [z/OS UNIX System Services User's Guide](#)
  - [z/OS MVS System Commands](#)
  - [z/OS MVS Diagnosis: Reference](#)
  - [z/OS MVS Diagnosis: Tools and Service Aids](#)
  - [z/OS MVS Initialization and Tuning Reference](#)
- z/OS UNIX External Website
  - <http://www-1.ibm.com/servers/eserver/zseries/zos/unix/>

---

The z/OS UNIX Tools and Toys page is linked off of the z/OS UNIX External Website:  
<http://www-03.ibm.com/servers/eserver/zseries/zos/unix/bpxa1toy.html>

---



# Questions?

---



**Session 21584**  
**Diagnosing Problems in a  
z/OS UNIX Environment:  
Operational PD**



# Appendices




# Appendix A: Signal Numbers

- Terminating Signals

SIGHUP	1	Hangup detected on controlling terminal
SIGINT	2	Interactive attention, attention key pressed
SIGABRT	3	Abnormal termination, raised by abort() function
SIGILL	4	Detection of an incorrect hardware instruction
SIGPOLL	5	Pollable event
SIGURG	6	High bandwidth data is available at a socket
SIGFPE	8	Erroneous arithmetic operation, i.e.. divide by zero
SIGKILL	9	Termination (cannot be caught or ignored)
SIGBUS	10	Bus error
SIGSEGV	11	Detection of an incorrect memory reference
SIGSYS	12	Bad system call
SIGPIPE	13	Write on a pipe with no readers
SIGALRM	14	Timer expired
SIGTERM	15	Termination (application)

---



- Terminating Signals (cont)

SIGUSR1	16	Reserved as application defined signal 1
SIGUSR2	17	Reserved as application defined signal 2
SIGABEND	18	Raised by the abend() function
SIGQUIT	24	Interactive termination
SIGTRAP	26	Communication signal used by the ptrace call
SIGXCPU	29	CPU time limit exceeded
SIGXFSZ	30	File size limit exceeded
SIGVTALRM	31	Virtual timer expired
SIGPROF	32	Profiling timer expired
SIGDANGER	33	Termination



- Job Control Signals

SIGSTOP	7	Stop (cannot be caught or ignored)
SIGCONT	19	Continue if stopped
SIGCHLD	20	Child process terminated, stopped, or continued
SIGTIN	21	Read from a control terminal attempted by a member of a background process group
SIGTOU	22	Write to a control terminal attempted by a member of a background process group
SIGTSTP	25	Interactive stop
SIGTHSTOP	34	Thread stop (cannot be caught or blocked or ignored)
SIGTHCONT	35	Thread continue (cannot be caught or blocked or ignored)



- Miscellaneous Signals

SIGIO	23	Completion of input or output
SIGIOER	27	Input or output error
SIGWINCH	28	Change size of window
SIGDCE	38	Reserved for exclusive use by DCE
SIGDUMP	39	Take a SYSMDUMP



---



## APPENDIX B: OMVS CTRACE to External Writer

- 1. Start an external writer for OMVS
  - **TRACE CT,WTRSTART=XWOMVS**
- 2. Turn on the OMVS CTRACE
  - **TRACE CT,ON,COMP=SYSOMVS**
  - **R xx,WTR=XWOMVS**
  - **R xx,OPTIONS=(ALL),END**
    - **JOBNAME** can be specified (“userid” associated with job/user; **OMVS** for kernel entries)
- 3. Stop the OMVS CTRACE
  - **TRACE CT,OFF,COMP=SYSOMVS**
- 4. Stop the external writer
  - **TRACE CT,WTRSTOP=XWOMVS**

For some problems, it’s necessary to collect more OMVS CTRACE data for a long period of time (possibly covering entire execution of a program or job). This is due to the limited size of OMVS CTRACE buffers in the SYSZBPXX dataspace.

## APPENDIX B:

# OMVS CTRACE to External Writer (cont)

- Sample cataloged procedure for external writer
  - The procedure places trace data on [three DASD data sets \(TRCOUTnn\)](#)
  - The procedure is placed in member XWOMVS of SYS1.PROCLIB
  - Update the DSNNAME and VOLSER as required for your installation
  - External writer for OMVS CTRACE should have [dispatching priority at least equal to OMVS](#)

```
//XWOMVS PROC
//IEFPROC EXEC PGM=ITTTTCWR,REGION=32M
//TRCOUT01 DD DSNNAME=SYS1.CTRACE1,VOL=SER=TRACE1,UNIT=DASD,
//          SPACE=(CYL,500),DISP=(NEW,KEEP),DSORG=PS
//TRCOUT02 DD DSNNAME=SYS1.CTRACE2,VOL=SER=TRACE2,UNIT=DASD,
//          SPACE=(CYL,500),DISP=(NEW,KEEP),DSORG=PS
//TRCOUT03 DD DSNNAME=SYS1.CTRACE3,VOL=SER=TRACE3,UNIT=DASD,
//          SPACE=(CYL,500),DISP=(NEW,KEEP),DSORG=PS
```

The dispatching priority of the external writer should have a dispatching priority at least equal to OMVS. Also, please give the external writer job as large a region as possible to minimize the chance of losing a buffer. The external writer will cut a LOGREC entry every time it fails to successfully write the buffer to the trace data set.

For more details of sending OMVS CTRACE records to an External Writer, refer to [z/OS MVS Diagnosis: Tools and Service Aids](#).

# APPENDIX C:

## Formatting OMVS CTRACE in IPCS

- IPCS 2.7.1.D (Analysis --> Traces --> Ctrace --> Display)

```
----- CTRACE DISPLAY PARAMETERS-----
COMMAND ==>

System      ==>          (System name or blank)
Component   ==> SYSOMVS  (Component name (required))
Subnames    ==>

GMT/LOCAL   ==> L          (G or L, GMT is default)
Start time  ==>          (mm/dd/yy, hh:mm:ss.dddddd or
Stop time   ==>          (mm/dd/yy, hh.mm.ss.dddddd)
Limit       ==> 0         Exception ==>
Report type ==> FULL      (SHORT, SUMMARY, FULL, TALLY)
User exit   ==>          (Exit program name)
Override source ==>
Options     ==> KERNINFO

To enter/verify required values, type any character
Entry IDs ==>  Jobnames ==>  ASIDs ==>  OPTIONS ==>  SUBS ==>

CTRACE COMP(SYSOMVS) LOCAL FULL OPTIONS((KERNINFO))

ENTER = update CTRACE definition.  END/PF3 = return to previous panel.
S = start CTRACE.  R = reset all fields.
```

OMVS CTRACE records can be formatted in IPCS via the IPCS panels (=2.7.1.D) or via the IPCS command line interface.

# APPENDIX C: OMVS CTRACE – Syscall Entry Record

FCN...open	SYSCALL...BPX1OPN		PID...010206A3	MODULE...BPXJPCPC
Z1	SYSCALL	0F080001	06:18:10.629194	STANDARD SYSCALL ENTRY TRACE
	ASID..032B	USERID...CSQ1BRK	STACK@....00000049	EF688000
	TCB...006C77B8	EUID.....00001077	PID.....010206A3	
+0000	00000026	00000000	D1C3E2C5	8C000185   .....JCSE...e
+0010	8000000C	00000000	2C84F44E	016BCC90   .....d4+...
+0020	00000000	40404040	00000048	338F6560   .....-
+0030	80000007	0000004E	61898982	A5F1F061   .....+/iibv10/
+0040	C3D6D4D7	61C3E2D8	F1C2D9D2	61998587   COMP/CSQ1BRK/reg
+0050	89A2A399	A861C3E2	D8F1C2D9	D261C3A4   istry/CSQ1BRK/Cu
+0060	99998595	A3E58599	A2899695	61C4E2D5   rrentVersion/DSN
+0070	6194987A	7AD8D4C7	00000402	000001B0   /mq:QMG.....
+0080	00000000	00000081	0594003D	00000049   .....a.m.....
+0090	EF6888B8	00000049	EF6888B0	00000049   ..h.....h....
+00A0	EF6888B4	14D12EC0		..h..J.{

- +18 RETURN ADDR TO SYSCALL CALLER (REG14)
- +2C ENTRY PARMLIST POINTER (REG 1)
- +30 NO OF PARMS IN PARMLIST
- +34 VARIABLE NUMBER OF PARMS (CONTENTS)

OMVS CTRACE syscall entry records can be used to identify the caller of a syscall (BPX1OPN open in this example), as well as the parmlist information for a syscall request. The first 4 bytes of each parameter will be traced, with the exception of pathname parameters, for which up to 64 bytes of the pathname will be traced.

# APPENDIX C: OMVS CTRACE – Syscall Exit Record

```
FCN...open          SYSCALL...BPX1OPN  PID...010206A3  MODULE...BPXJCPC
Z1      SYSCALL    0F080002  06:18:10.629372  STANDARD SYSCALL EXIT TRACE
      ASID..032B    USERID....CSQ1BRK  STACK@....00000049  EF688000
      TCB...006C77B8  EUID.....00001077  PID.....010206A3
+0000  00000026  00000000  D1C3E2C5  8C000000  | .....JCSE.... |
+0010  8000000A  00000000  FFFFFFFF  00000081  | .....a |
+0020  0594003D          | .m.. |
```

- **+18** RETURN VALUE (RV)
- **+1C** RETURN CODE (ERRNO)
- **+20** REASON CODE (ERRNOJR)

OMVS CTRACE syscall exit records can be used to identify the success of a syscall request. In this example, the BPX1OPN open syscall request failed with rc81 (ENOENT) and rsn0594003D. Per BPXMTEXT, rsn0594003D indicates JRDirNotFound.