MVS Open and Distributed Project Opening and What's New with OS/390 UNIX Session 2904

March 6, 2000

Garth Godfrey
OS/390 UNIX Systems Services Development
IBM Poughkeepsie, NY
ggodfrey@us.ibm.com
http://www.ibm.com/s390/unix

Session Objectives

- At the completion of this session, you will be able to:
  - Identify OS/390 UNIX enhancements in Releases 8 and 9
    - Shells & Utilities
    - File System
    - Kernel
  - State how the enhancements can be used to benefit your installation

Overview - OS/390 V2R8 UNIX

Need Addressed	Solution
Provide more function available on other UNIX platforms	Magic number (#!) support Shell support for [[...]] Shell support for autoloaded functions
Interoperability with MVS facilities.	cp, mv, pax, tar support of MVS dsn tso command keywords
Better security and system management	Superuser Granularity User resource limits Surrogate userid vi temporary files skulker (soon) SETOMVS command options
Better diagnostic tools	BPXBATCH return codes dbx enhancements

OS/390 V2R8 Magic Number

- OS/390 UNIX now supports script files that start with a line:

  `#! /usr/bin/shell`

  - where ***`/usr/bin/shell`*** is the pathname of the command interpreter to use to execute the script
  - e.g. If you installed Perl in /usr/bin/Perl, start your Perl scripts with:
    #! /usr/bin/Perl

  - APAR back to R7 --- OW39467

OS/390 V2R8 Shell Enhancements

- **[[** *test_expr* **]]**
  - Korn shell (ksh) syntax used for conditional tests
  - Reserved word command used frequently in ksh scripts for flow control

    if [[-e file]]; then ...

- **Autoloaded functions**
  - Shell function definitions are "loaded" (stored internally by the shell)
  - Autoloaded functions are "loaded" from a directory the first time they are called

Double square brackets syntax is widely used in Korn shell scripts
These scripts are often part of UNIX applications
Reserved word commands are built into the shell
[[...]] can be used anywhere a shell command can be used
Shell functions are sequences of shell commands (like shell scripts)
Functions are stored in the shell, so they are faster than shell scripts
Typically, a user's .profile sets up ENV=setup , and the setup file contains all function definitions the user wants available from the shell command line
This must be processed for login shells, and child shells running shell scripts and background jobs
Compatible with Korn shell

Autoloaded function setup

- Identify functions as autoloaded (undefined) in user's ENV setup script
  - autoload func1 func2 func3 ...
- Put function definitions in directories
  - administrators may define shared directories with common functions
- Define the FPATH shell variable to specify the search path for functions

autoload is an alias for "typeset -fu"

1st step is optional, but recommended for slightly better performance

FPATH has the same format as the PATH variable
FPATH=dir1:dir2:dir3

cp / mv support

- copy / move between:
  - UNIX file <=> MVS seq ds
  - UNIX files <=> MVS PDS or PDSE members
  - MVS PDS or PDSE => UNIX dir
  - MVS PDS or PDSE member => MVS PDS or PDSE
- **cp**/**mv** can create an MVS sequential dataset
- support for text / binary / executables
- syntax examples:

```
cp unixfile "//'godfrey.file1'"
cp f1 f2 f3 "//'posix.godfrey.lib'"
```

To copy/move to a PDS or PDSE, the dataset must be allocated before the copy/move. Examines file attributes to pick text/bin/exec, or user can override with option.
-P can be used to specify the fopen() parms used to create the seq. ds, including RECFM, SPACE, LRECL, BLKSIZE.
Example without quotes: //mylib\(pgm1\)
Compatibility issue: UNIX scripts that use cp /$x/file may resolve to //file. Was treated as a /file.Now treated as an MVS dataset (Portable scripts ensure ///file).
Considering adding a shell var for compatibility.

pax/tar support of MVS datasets

- MVS datasets are only supported as the **archive** file
  - You cannot store MVS dataset members as component files within an archive.
  - You cannot extract component files from an archive directly into an MVS dataset
- Syntax:

```
pax -wf "//'user.lib(archive)'" *
pax -rf "//'user.lib(archive)'"
```

pax and tar package HFS directory trees into a single file called an "archive". Archives can be moved to other systems or directories.
"Extracting files" means moving files from the archive into HFS files.
Avoids OPUT of archives from MVS ds to UNIX, and OGET of archives from UNIX to MVS ds. (Often problems with forgetting to use binary, or space in UNIX filesystem.) Archives are always treated as binary
When writing to a PDS member, the PDS must already exist
pax will automatically overwrite an existing archive (consistent with UNIX)

tso shell command

- New allocation keywords
  - PATH('*pathname*') file in the HFS
  - PATHOPTS(*pathopt*[,*pathopt*]...) list of path options
  - PATHMODE(*pathmode*[,*pathmode*]...) list of modes (permissions)
  - PATHDISP(KEEP|DELETE[,KEEP|DELETE]) normal, abnormal file disposition
  - FILEDATA(TEXT|BINARY) data text or binary

Superuser Granularity

UNIXPRIV Resource Names Supported:

- CHOWN.UNRESTRICTED
- SUPERUSER.FILESYS
- SUPERUSER.FILESYS.CHOWN
- SUPERUSER.FILESYS.MOUNT
- SUPERUSER.IPC.RMID
- SUPERUSER.FILESYS.PFSCTL
- SUPERUSER.PROCESS.GETPSENT
- SUPERUSER.PROCESS.KILL
- SUPERUSER.PROCESS.PTRACE
- SUPERUSER.SETPRIORITY
- SUPERUSER.FILESYS.VREGISTER

- CHOWN.UNRESTRICTED - Existence of profile allows user to change ownership of his own file.
- SUPERUSER.FILESYS - READ (or higher) allows a user to read any HFS file and read/ search any HFS directory. UPDATE (or higher) allows a user to write to any HFS file. CONTROL (or higher) allows a user to write to any directory
- SUPERUSER.FILESYS.CHOWN - Allows user to change ownership of any file
- SUPERUSER.FILESYS.MOUNT - Allows user to issue mount, unmount, quiesce, and unquiesce requests. READ permission to mount NOSETUID. UPDATE permission: full function mount points.
- SUPERUSER.IPC.RMID - Allows user to do ipcrm calls to clean up leftover IPC mechanisms
- SUPERUSER.FILESYS.PFSCTL - Allows user to call pfsctl()
- SUPERUSER.PROCESS.GETPSENT - Allows user to see all processes with getpsent() (ps command)
- SUPERUSER.PROCESS.KILL - Allows user to send signals to any process.
- SUPERUSER.PROCESS.PTRACE - Allows user to dbx (ptrace) any process.
- SUPERUSER.SETPRIORITY - Allows user to increase his priority. (setpriority(), nice())
- SUPERUSER.FILESYS.VREGISTER - Allows user to issue vregister() to register as a vfs file server

OS/390 UNIX User Limits

- Stored in OMVS segment of user profile
  - CPUTIMEMAX
  - ASSIZEMAX
  - FILEPROCMAX
  - PROCUSERMAX
  - THREADSMAX
  - MMAPAREAMAX
- `ADDUSER ... OMVS(CPU(100) ASSIZEMAX(200M) ...)`

Up until this release, limits specified in BPXPRMxx applied to all processes. If one user needed a particularly large region, lots of CPU time, 100's of processes or many threads, it was necessary to set the system limit high enough for the largest user.
The BPXPRMxx system limits still apply, unless there is a specific limit value specified in the OMVS segment for the user.
If the user has any of the limits (FILEPROCMAX, PROCUSERMAX, THREADSMAX or MMAPAREAMAX) specified, these limits apply.
For MAXCPUTIME and MAXASSIZE, the limit is environment sensitive. For example, if the address space is TSO or batch, then a region size and time limit are already established and are not changed. Once dubbed, these limit values become the new hard limit, which would allow the user to invoke the setrlimit function (or the ulimit command) to raise their limit.
For the setuid/exec case which is typical of daemons (inetd, rlogin, telnet), the address space will take on the new region size and CPU time limit when the exec function completes.
You may be wondering why the RACF keywords are CPUTIMEMAX instead of matching (MAXCPUTIME) the name of the limit in BPXPRMxx. The reason is to allow abbreviations of the shortest possible length within the RACF commands that define these keywords.

Surrogate Userid

- **Switch to userid without password**
- **SETROPTS CLASSACT(SURROGAT)**
- **RDEFINE SURROGAT BPX.SRV.JACK UACC(NONE)**
- **PERMIT BPX.SRV.JACK CLASS(SURROGAT) ID(JILL) ACCESS(READ)**
- **SETROPTS RACLIST(SURROGAT) REFRESH**

JILL running shell
- ►su jack
- ►FSUM5019 Enter the password for jack: <enter>
  - new shell started with userid JACK
- ►su -s jack
  - new shell started with userid JACK

- Support in setuid, seteuid, and __passwd to honor SURROGAT

In the past, BPX.SUPERUSER FACILITY class profile that allowed UNIX users to switch to superuser auhority without having to share the password of some root userid.
The ability to have similar processing between different userids has been added via the use of the SURROGAT class profiles. In this example, JILL has been given surrogate authority to JACK. This allows JILL to enter the su shell command to enter a new shell running under JACK without having to provide a password.

vi editor temporary files

- **TMP_VI**
  - New environment variable
  - Location of vi editor temporary files available for recovery
  - If /tmp is TFS,
    - TMP_VI=/var in /etc/profile
    - TMP_VI=/var exrecover in /etc/rc (run at IPL)
- **set autoflush=n**
  - n = number of seconds to flush buffers to temporary files
  - minimizes loss of vi editing when system crash

TMP_VI Contains a directory pathname that can be specified by an administrator as a location for vi's temporary files. This is useful if the current default directory for these files (usually /tmp) is implemented as a TFS. In this case, all vi's temporary files that the exrecover daemon uses for recovery would be gone after a system crash.
We recommend that this environment variable be set by a system administrator as opposed to a user setting it for their environment. The exrecover command run during IPL must use the same directory as the user's vi sessions.

skulker - new shell command

- Coming soon via APAR OW42841
- **skulker** [-irw] [-l *logfile*] *directory days_old*
- removes files in directory older than specified number of days
- shell script in /samples
  - copy to /bin/skulker or /usr/sbin/skulker
  - can be modified by installation
  - Protect it from hackers! (non-writable)
- example:
  skulker /tmp/ 100
- use cron to schedule it to run regularly

IPL Avoidance SETOMVS

- Adjust HFS and socket parms dynamically
  - FILESYSTYPE
  - NETWORK
  - SUBFILESYSTYPE
- **Can IPL in min mode, then add HFS FILESYSTYPE and mount a new ROOT.**
- **Can add a new TCP/IP stack.**

SETOMVS RESET=xx Operator cmd

BPXPRMxx

Over time, our goal is to allow you to dynamically modify anything in the BPXPRMxx parmlib member and have it take effect immediately. We are biting off pieces at a time. This round was done to allow installations to add new FILESYSTYPEs after IPL. This would most likely be used to add something like NFS after the system is up and running.
One can also use it to add new TCP/IP stacks.

BPXBATCH / dbx

- **BPXBATCH** passes back ending status from requested program
- **dbx**: new subcommands
  - **args** displays the argument count and a list of arguments that are passed to the user's program
  - **listfiles** displays the list of files associated with each module in the load map
  - **listfuncs** displays a list of functions associated with each file in the program
  - **record** appends the user's command lines to the specified file
- **dbx**: new -f command option
  - allows dbx to run in super lazy read mode, which speeds up the reading of symbols for large programs and DLLs.

BPXBATCH can be run in 2 ways. It can be run from JCL, in which case, no change to return code processing occurs.
The other way it can be invoked is via calls from TSO, CLISTS, REXX, or any other program. In these cases, the old way was for BPXBATCH to pass back a return code saying how well BPXBATCH did. The program which was run could have failed miserably, but BPXBATCH was successful (RC=0).
The logic is changed to pass back the ending status of the program which BPXBATCH ran.

Overview - OS/390 V2R9 UNIX

Need Addressed	Solution
Provide more function available on other UNIX platforms	tcsh C-shell fuser Shared library support Queued signals
Interoperability with MVS facilities.	sysvar Shared HFS in a SYSPLEX Megabyte Map services
Better system management	D OMVS operands BPXBATSL pax / tar support long link names PFS recycle
Better diagnostic tools	BPXPRMxx syntax checker JOBLOG to stderr dbx enhancements

Numerous customer requests have been made for these functions to improve the portability of applications, shell scripts, and users from other UNIX platforms

C-shell, fuser, and shared libraries are typically provided by other UNIX platforms. The following utilities provide shared library support:

- extattr
- ls
- cp
- mv
- pax
- tar

sysvar provides shell script access to an MVS system variable

pax/tar change solves a long-standing UNIX restriction.

tcsh - An enhanced C-shell

- **Superset of BSD C-shell function**
- **Advantages**
  - Portability of C-shell scripts from other UNIX platforms
  - Interactive C-shell users get familiar look and feel, including:
    - **command-line editor**
    - **programmable word-completion**
    - **spelling correction**

Numerous customers and vendors have applications with C-shell scripts, and/or have users familiar with a C-shell.

Since Berkeley source code has been freely available, there are numerous versions of C-shell in use. **tcsh** is based on the Berkeley C-shell, but is legally "clean", and is freely available from the internet. We've invested a lot in porting this code, customizing it for OS/390 and extensive testing. It is fully-supported through the normal IBM service channels.

It is completely compatible with the Berkeley C-shell, with enhancements that provide advantages for interactive use.

Many customers have requested **tcsh** by name.

tcsh word completion

- tcsh completes words anywhere in the line, given a unique abbreviation for filenames, commands, and variables.
- filename examples:

```
> ls
j           k           l           ldir        lost+found  m
> ls -l lost<TAB>
> ls -l lost+found
-rw-rw-rw-   1 WELLIEA  SYS1           5 Sep 24 14:42 lost+found
> ls -l ld<TAB>
> ls -l ldir/
total 24
-rw-rw-rw-   1 WELLIEA  SYS1           5 Sep 24 14:52 file1
-rw-rw-rw-   1 WELLIEA  SYS1           4 Sep 24 14:52 file2
-rw-rw-rw-   1 WELLIEA  SYS1          13 Sep 24 14:52 file3
```

- command examples:

```
> pass<TAB>
> passwd

> ggg<TAB>
> ggg-ridiculously-long-command
This command does nothing but print this message.
```

A word is basically any set of characters separated from other characters by whitespace (blank, tab) or shell special characters.

The shell treats words as commands at the beginning of a shell line or following `;', `|', `|&', `&&' or `||'

The shell treats words as variables when they start with $

All other words are completed as filenames

In the examples, **lost** is replaced with **lost+found** on the command input line
ld is replaced with **ldir/** **/** put after a directory name

Note: sh also supports filename completion within command-line-editing
Esc\ in vi mode

Command examples - good for commonly misspelled commands and long command names that start with something unique

tcsh word completion listing

- tcsh lists possible word completions when Ctrl-D is typed
- filename example:

```
> ls
j           k           l           ldir        lost+found  m
> ls l<^D>
l       ldir/     lost+found
> ls l
```

- command example:

```
> com<^D>
comm    command   complete  compress
> com
```

Word completion listing shows possible completions, but does not change the command input line.

Command completion listing uses the value of the **path** shell variable.

Spelling correction prompts:

y or space = execute the corrected line
n = execute the line unchanged
e = leave the uncorrected command in the input buffer
a = abort (like ^C)

tcsh word completion

- tcsh has lots of shell variables and editor commands to customize word completion
- The **complete** built-in command can be used to customize completion and spelling correction.

The ability to custom tailor the completions via user complete commands is one of the strengths of tcsh.

IBM ships a /samples/complete.tcsh (to be copied over to /etc/complete.tcsh) which is customized for many of the OS/390 UNIX shell commands.

shell feature comparison

	OS/390 sh	tcsh
Job control	Y	Y
Aliases	Y	Y
Shell functions	Y	N
Full I/O redirection	Y	N
Directory stack	N	Y
Command History	Y	Y
Command-line editing	Y	Y
Rebindable command-line editing	N	Y
User name lookup	Y	Y
Filename completion	Y	Y
History completion	N	Y
Programmable completion	N	Y
Periodic command execution	N	Y
Spelling correction	N	Y
User-specifiable startup file	Y	N
Full signal trap handling	Y	N

While this is not a complete chart of shell features, it is intended to show that both sh and tcsh have a rich set of function.

sh (which includes many Korn shell features) is generally superior for programming shell scripts.

tcsh has some powerful unique features for interactive use.

tcsh files

File	Description	Samples
/etc/csh.cshrc	System-wide .cshrc	Yes
/etc/csh.login	System-wide .login (only for login tcsh shell)	Yes
$HOME/.tcshrc	User's setup. Read instead of $HOME/.cshrc if it exists (tcsh only)	Yes
$HOME/.cshrc	User's setup. Read at startup (if no .tcshrc)	No
$HOME/.history	Read by login shells at startup to initialize the history list	No
$HOME/.login	Read by login shells at startup	Yes
$HOME/.cshdirs	Read by login shells at startup to initialize the directory stack	No
/etc/complete.tcsh	Completion rules. Read by $HOME/.tcshrc (in sample)	Yes
/etc/csh.logout	System-wide .logout	No
$HOME/.logout	Read by login shells at termination	No

The tc shell reads one or more files (if they exist) when it is invoked (and before displaying the first prompt). These files collectively alter:
- terminal settings
- variable values
- aliases
- key bindings for the command-line editor
- programmed completions

It also provides the ability to perform actions at logout time.

The /samples files can be modified OR
profiles from a UNIX platform can be converted to EBCDIC & modified.

fuser - new shell command

- fuser - List process IDs of processes with open files
  - UNIX98 conformant
  - Useful for finding the current users of a file, or a filesystem (e.g. before unmount)
- **fuser** [-cfku] *file ...*
  - -c report on filesystem containing file
  - -k kills the process (superuser only)
  - -u user name

UNIX98 (Single UNIX Specification, V2) is the industry-wide specification of common UNIX interfaces

fuser writes to standard output the process IDs of processes running on the local system that have one or more named files open.

Additional information (e.g. pathname, user names) is written to standard error.

-f is the default (report on only the specified files)

-k is an extension to the UNIX98 spec
allowed for users with READ access to SUPERUSER.PROCESS.KILL facility class profile

fuser

- examples:
  - `# fuser -u /g/welliea/file.c`
    `/g/welliea/file.c: 50331660(WELLIEA)`
  - `# fuser -u /g/welliea`
    `/g/welliea: 83886092c(WELLIEA)`
    `c` indicates in use as cwd
  - `# fuser -cu /usr/lpp/dfs`
    `/usr/lpp: 50331655(BPXROOT)`
  - `# fuser -uk /g/welliea/junk`
    `/g/welliea/junk: 33554447(WELLIEA)`
    `k` kills the process!

Examples assume that stdout and stderr are both directed to the terminal.

c following the pid indicates the file is a user's current working directory (cwd)
r would indicate a user's root

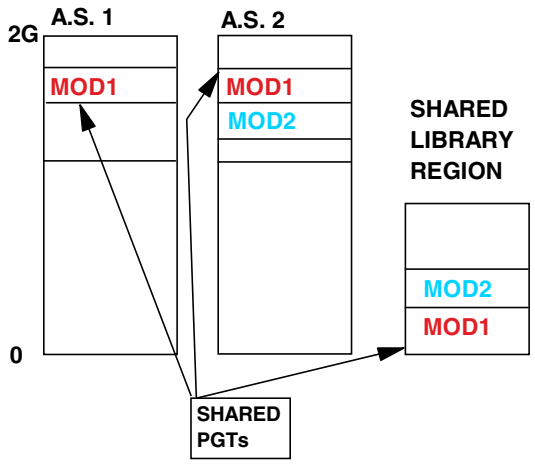
-c option is useful when you want to unmount a filesystem

Shared Libraries

- Support Shared Library Objects in a manner consistent with other UNIX implementations
  - Used by multi-process server applications
- Optimizes Sharing of HFS programs across the system in two flavors:
  - System Shared Libraries most optimal for heavy sharing of large HFS executables
  - User Shared Libraries most optimal for moderate sharing of smaller HFS executables

Shared Library support provides for optimal sharing of HFS executables without requiring the usage of LPA or CSA for the module storage, as is the case today. Any DLL that is loaded from a C program, for example, can be identified as a shared library program.
Two different types of Shared Library Programs are supported. System Shared Library Programs are intended for large executables that are shared by many address spaces across the system. User Shared Library Programs are intended for smaller programs that are shared by a smaller set of user address spaces.

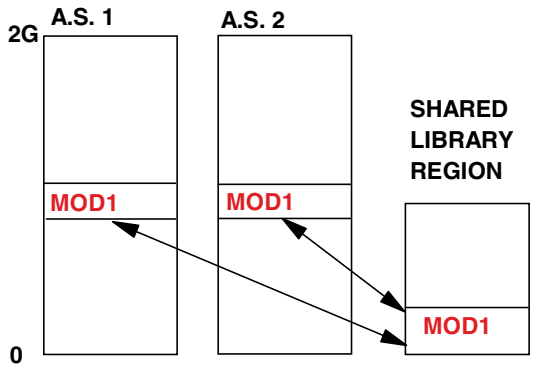
System Shared libraries



- ★ **Intended for Large HFS Executables**
- ★ **Modules Shared on a megabyte boundary**
- ★ **Maps into high end private storage**

System Shared Library Programs are the most optimal way to share large HFS executables across many address spaces in the system.
These executables are shared on a megabyte boundary to allow for the sharing of a single page table (similar to LPA). A mapping is established between the using address space and the Shared Library Region that the kernel maintains. The storage used in the user address space to establish the mapping is from the high end of private storage so that it does not interfere with the virtual storage used by the application program. To tag a HFS executable file as a System Shared Library Program you need to turn on a new extended attribute for the file. Using the shell command extattr +l fn to turn it on requires permission to BPX.FILEATTR.SHARELIB. extattr -l fn or file modification turns off the attribute.
Once a HFS executable is loaded into the shared library region reloads of the executable will see improved performance due to the elimination of I/O.

User Shared libraries



- ★ **Intended for smaller HFS Executables**
  - ▸ **DLLs with .so suffix**
- ★ **Modules Shared on a page boundary**
- ★ **Mapped into low end storage in the user region**

User shared libraries allow anyone to define and use shared libraries up to a system limit. This level of sharing does not use the shared page table approach, which causes it to consume ESQA to create the sharing of module storage. User shared libraries are shared on a page boundary. User shared library programs are identified by the .so suffix in the file name. When a particular DLL with the .so suffix is loaded from a C program the DLL is treated as a user shared library program. If the user shared library program is already loaded into the Shared Library Region, a mapping is established between the user address space and the Shared Library Region copy of the module without requiring the actual load of the program.

Shared libraries - new BPXPRMxx statements

- SHRLIBRGNSIZE()

  limits the size of the **system** shared library region
  - range 16MB - 1.5GB
  - default value - 64MB

- SHRLIBMAXPAGES()

  Limits the amount of data space storage pages that can be allocated for **user** shared library modules
  - range 1-16777216
  - default value - 4096

Shared libraries - shell commands

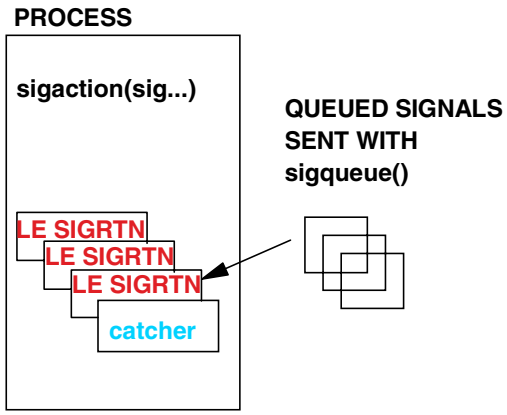
- new extended attribute for executable files to be loaded from the system shared library region
- **extattr** +l sets the sharedlib attribute (BPX.FILEATTR.SHARELIB)
  -l turns it off
- **ls -E** displays the extended attributes (now 4 characters)

```
$ ls -E pgm*
 -rwxr-xr-x  --s-  1 WELLIEA  SYS1    794624 Sep 26 22:30 pgmo
 -rwxr-xr-x  -p--  1 WELLIEA  SYS1    802816 Sep 23 10:00 pgmp
 -rwxr-xr-x  ---l  1 WELLIEA  SYS1     73728 Aug 24 12:03 pgmshr
```

- **cp -p** or **mv** preserves the extended attributes
- **pax -pe** (or **-px**) preserves the extended attributes
  -E displays ext attr
- **tar -p** preserves the extended attributes
  -E displays ext attr

The shared library region attribute is treated like other extended attributes in S&U

Queued signals



- sigqueue()
  - UNIX98 conformant
  - APARs made available on R6, R7, R8
- BPXPRMxx statement (or SETOMVS)
  - MAXQUEUEDSIGS
    - maximum number of signals that can be concurrently queued within a single process (1 - 100000)

Sigqueue() is a UNIX 98 function which allows a process to send a signal which will be queued to the target. In the past, multiple signals of the same signal number would result in a single delivery of that signal. With sigqueue, the signal will be delivered as many times as it is sent. Sigqueue allows the caller to provide 1 word of data to pass to the target. This could be an address in shared memory.
This service is useful to applications that want to be asynchronously notified of an event, for example the completion of an I/O request (Lotus R5 uses this)
A new keyword is added to BPXPRMxx to allow the installation to set a limit on the number of signals that can be queued.

sysvar - new shell command

- Display static system symbols
- **sysvar** *var*
  - obtain the value of a system variable defined in IEASYMxx or IPL parms
  - output to stdout may be easily captured in a shell variable for use in a shell script

- examples:
  - ```
    sysvar SYSNAME
    MVS1
    ```

  - ```
    system_name=$(sysvar SYSNAME)
    echo "Creating logfile $system_name.logfile"
    create_log_file >$system_name.logfile
    ```

Allows shell scripts to be used without change on different systems.

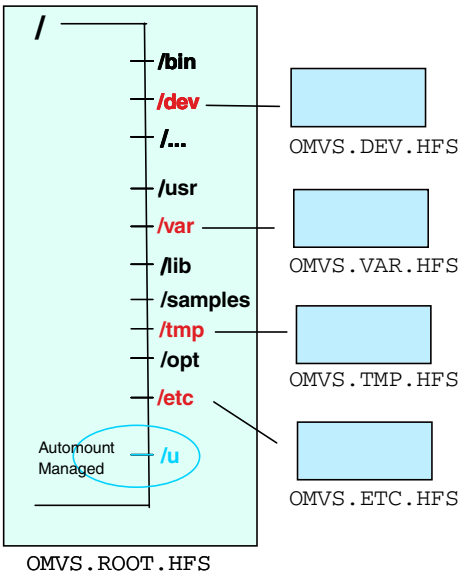
2nd example shows this.

Shared HFS in a Sysplex

- Allow R/W data to be shared across the sysplex the way R/O data is shared today.
- Utilize XCF messaging services to communicate data from system to system.
- Couple Data set for Sysplex wide mount table & serialization
- SETOMVS support for manipulating the file hierarchy in the sysplex.
- Recovery for transferring dead system file systems to another server

Today, file systems can be mounted Read Only to all systems in the sysplex. In R9 a R/W file system is accesssible to all systems in a sysplex. XCF message services are used to communicate between systems in the sysplex. This allows meta data such as mount structure, for example, to be maintained in a couple data set to allow each system to know where it must communicate to access a file.
Recovery is provided, such that, If one system fails, another system in the sysplex will take over the file sharing responsibilities for the failing system. This will be invisible to the application

Scenario 1- Single Standalone System, Today



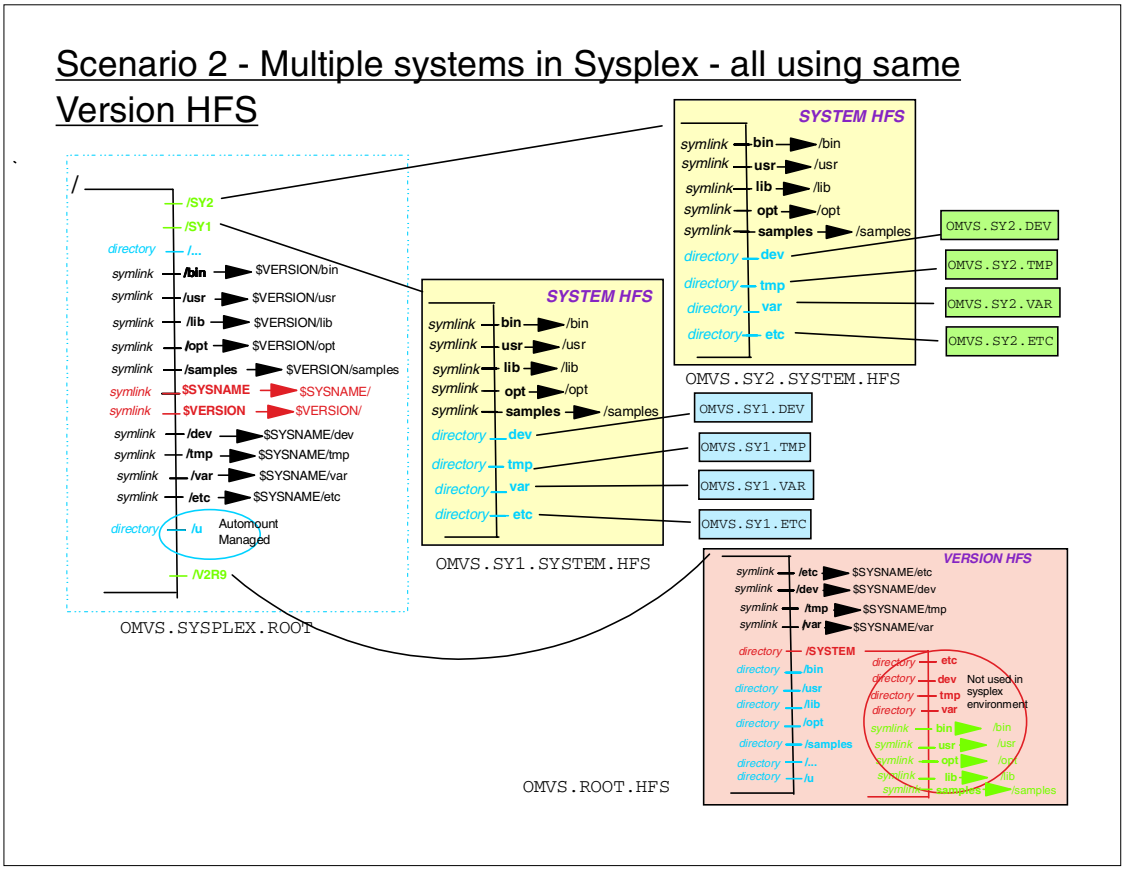
BPXPRMxx

```
ROOT
FILESYSTEM('OMVS.ROOT.HFS')
TYPE (HFS) MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.DEV.HFS')
TYPE(HFS) MODE(RDWR)
MOUNTPOINT('/dev')

.
.
.
```

The directories in the Root HFS represent the "first level" directories created by IBM.
The directories in Red are used as mountpoints for other HFS data sets.

Today, each system defines via Parmlib its own standalone HFS infrastructure. IBM provides "first level" directories in the root HFS data set for directories such as /bin, /usr, etc.. Mount points are set up for directories /dev, /var, /tmp and /etc.

Scenario 2 - Multiple systems in Sysplex - all using same Version HFS



In this scenario, systems SY1 and SY2 have their own System HFS data sets defined with symbolic links into the Version HFS for V2.R9. For example, when a reference is made from system SY1 for the bin directory it actually causes a reference in the Version HFS data set that is shared across the Sysplex. Any changes to this directory caused by system SY1 will be reflected to the other systems in the Sysplex.

Shared HFS in a SYSPLEX

- New sample jobs to create new HFS structures
- BPXPRMxx parameters
  - SYSPLEX(YES|NO)
  - VERSION('nnnn')
  - ROOT and MOUNT ... new keywords
    - SYSNAME(sysname)
    - AUTOMOVE|NOAUTOMOVE
- TSO MOUNT command
    - SYSNAME(sysname)
    - AUTOMOVE|NOAUTOMOVE

SYSPLEX(YES) indicates whether a system is to be initialized in a sysplex environment or operate in local mode. The first system entering the sysplex with SYSPLEX(yes) initializes a Couple Data Set (CDS), which controls shared HFS mounts. The value of this parameter cannot be changed dynamically.
VERSION('nnnn') indicates the release or version of root HFS.
New BPXPRMxx optional keywords on the ROOT and MOUNT parameters: SYSNAME(sysname) and AUTOMOVE|NOAUTOMOVE. .
On the ROOT parameter, SYSNAME(sysname) is the name of a system in a sysplex that was IPLed with SYSPLEX(YES)
AUTOMOVE|NOAUTOMOVE parameters indicate whether, if the specified root file system owner goes down, the root file system can be automatically moved to another system, which then becomes the owner for that root.
On the MOUNT parameter, SYSNAME(sysname) specifies the particular system on which a mount should be performed.
SYSNAME(system_name) specifies the specific system on which a mount should be performed (this system must be IPLed with SYSPLEX(YES)).
AUTOMOVE|NOAUTOMOVE keywords indicate whether the ownership of a file system is to be transferred if the file system's owner goes down.

Shared HFS - shell commands

- **df -v** display new filesys fields
  - file system ID (owner/mounted file system server)
  - file system ID issuing a quiesce request
- New shell commands in /usr/sbin
  - **mount** - mounts a file system or lists all mounts over a file system
  - **chmount -** changes the mount attributes of a specified file system in a sysplex
  - **unmount** - removes file systems from the file hierarchy
- Require UID 0 or READ access to SUPERUSER.FILESYS.MOUNT resource in the UNIXPRIV class

mount [-t fstype] [-o fsoptions] [-d destsys [-a yes|no]
[-s nosecurity|nosetuid] -f fsname pathname
mount -q [-v] [-d destsys] pathname...
chmount [-DR] [-d sysname] [-a yes|no] path
unmount [-Rv] [-o unmount_option] pathname

Shared HFS Sessions

- Session 2927 OS/390 UNIX Filesystem Sysplex Support (Shared HFS) - Part 1
  - Marriott - Ground Level - Grand Ballroom Salon E
  - 03/09/2000 (Thu) - 1:30 PM
- Session 2928 - Part 2
  - 03/09/2000 (Thu) - 3:00 PM

Megabyte Map Services

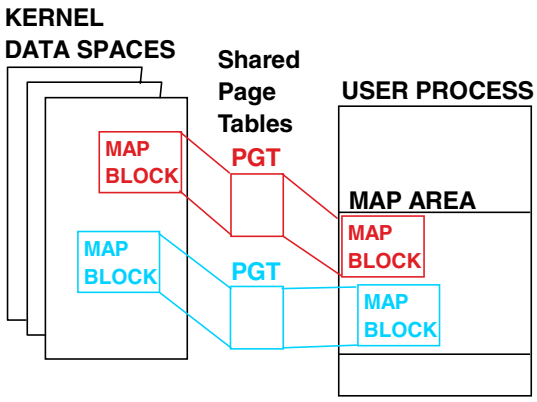
- Map blocks >2Gig of virtual
- Connects done using shared page tables.
- Fast access to large virtual

- __map_init()
  - Define Map Area size
  - Define size of Map Blocks
  - Map Area recreated on fork
  - Map cleanup when caller terminates
  - Must have BPX.MAP FACILITY class access

- __map_service()
  - Connect New/Old block to map area
  - Disconnect a block from map area
  - Delete a map block
  - Change access R/O or R/W

The map services give the ability to manage greater than 2 gig of virtual and selective access blocks of this storage from multiple address spaces.
A first process does a call to __map_init(). This process must be superuser or permitted to BPX.MAP FACILITY class profile. This process then can do fork and each child gets its own map area at the same virtual address as the parent.
Any of these processes can now do __map_service calls.
A CONNECT request will map a block of virtual in the caller's map area to a block of storage in a kernel data space and return a token for this block of storage. Other processes can connect to this same block of storage in other address spaces. The intent is to always connect a block at the same virtual address in the map area so that pointers can be used.
Any virtual in the map area that is not connected to a block is hidden (0C4 is read or write).
Applications can keep many gigabytes of virtual in the map blocks and only connect to what they need. The connection process involves manipulation of the user segment table to point to shared page tables, thus avoiding any movement of data.

Megabyte Map services



__map_init
map area recreated on fork

__map_service
Process term of __map_init process cleans up the map.

D OMVS,PFS

- Enhance DISPLAY OMVS to show the current PFS configuration.
  - FILESYSTYPE
  - SUBFILESYSTYPE
  - NETWORK
  - Common INET Routing tables

There was no way to display the current configuration of started Physical File Systems, i.e. HFS, TCP/IP, NFS Client, etc.

Support in Rel 8 for dynamically starting PFSes via SETOMVS RESET= made it harder to keep track of what was started.

D OMVS,PFS

- A typical display looks like:

```
d omvs,p
BPXO046I 10.15.15 DISPLAY OMVS 135
OMVS     000E ACTIVE          OMVS=(33)
PFS CONFIGURATION INFORMATION
 PFS TYPE     DESCRIPTION            ENTRY      MAXSOCK OPNSOCK
  TCP          SOCKETS AF_INET        EZBPFINI    50000     234
  UDS          SOCKETS AF_UNIX        BPXTUINT       64       0
  NFS          REMOTE FILE SYSTEM     GFSCINIT
  TFS          LOCAL FILE SYSTEM      BPXTFS
  HFS          LOCAL FILE SYSTEM      GFUAINIT
  BPXFTCLN     CLEANUP DAEMON         BPXFTCLN
  BPXFTSYN     SYNC DAEMON            BPXFTSYN
  BPXFPINT     PIPES                  BPXFPINT
  BPXFCSIN     CHARACTER SPECIAL      BPXFCSIN

 PFS TYPE     PARAMETER INFORMATION
  NFS          biod(3)
  HFS          CURRENT VALUES: FIXED(0) VIRTURAL(3)
```

D OMVS,PFS

- A typical display with Cinet looks like:

```
d omvs,p
BPXO046I 11.33.59 DISPLAY OMVS 561
OMVS     0045 ACTIVE          OMVS=(CI)
PFS CONFIGURATION INFORMATION
 PFS TYPE     DESCRIPTION            ENTRY     MAXSOCK OPNSOCK
  CINET        SOCKETS AF_INET        BPXTCINT   64000        2
  IBMUDS       SOCKETS AF_UNIX        BPXTUINT      64        0
  HFS          LOCAL FILE SYSTEM      GFUAINIT
  BPXFTCLN     CLEANUP DAEMON         BPXFTCLN
  BPXFTSYN     SYNC DAEMON            BPXFTSYN
  BPXFPINT     PIPES                  BPXFPINT
  BPXFCSIN     CHARACTER SPECIAL      BPXFCSIN

 PFS NAME     DESCRIPTION            ENTRY      STATUS  FLAGS
  TCPIP        SOCKETS                EZBPFINI   ACT     SC
  TCP2         SOCKETS                EZBPFINI   INACT
  ANYNET       SOCKETS                ISTOEPIT   INACT

 PFS TYPE     PARAMETER INFORMATION
  HFS          CURRENT VALUES: FIXED(0) VIRTUAL(32)
```

D OMVS,CINET

- The Cinet display shows the routing tables that are used to select a stack for outbound data and connections.

```
d omvs,cinet
BPXO047I 15.23.12 DISPLAY OMVS 956
OMVS     0045 ACTIVE          OMVS=(CI)
HOME INTERFACE INFORMATION
 TP NAME    HOME ADDRESS      FLAGS
  TCPIP      127.116.117.233   DRS
  TCPIP      127.116.118.234
  TCP2       127.116.119.235

HOST ROUTE INFORMATION
 TP NAME    HOST DESTINATION
  TCPIP      127.117.194.234
  TCPIP      127.117.195.234

NETWORK ROUTE INFORMATION
 TP NAME    NET DESTINATION   NET MASK          METRIC
  TCPIP      127.111.000.000   255.255.000.000    10
  TCPIP      127.113.000.000   255.255.000.000     0
  TCPIP      197.119.119.000   255.255.255.000     F
  TCP2       009.000.000.000   255.000.000.000     F
```

BPXBATSL

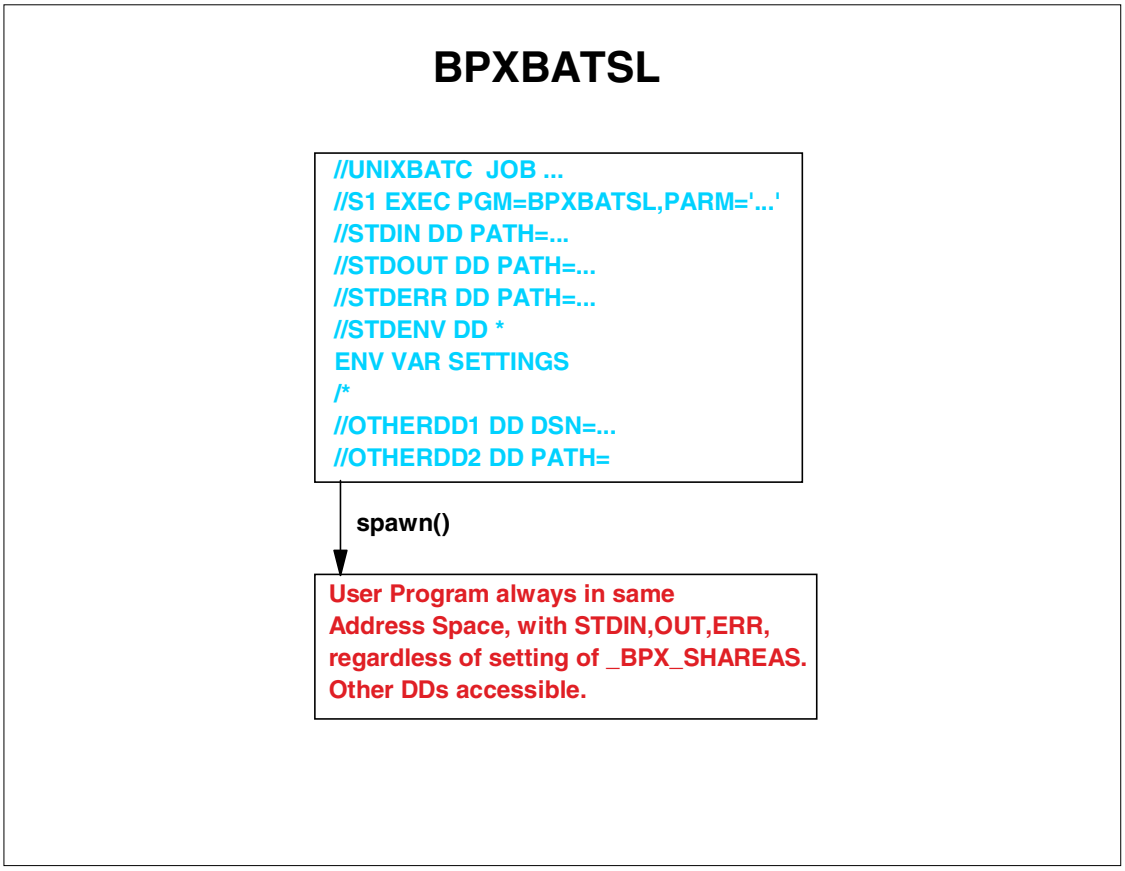
New entry point into BPXBATCH that will always spawn within the BPXBATCH Address Space

Invokes spawn with the "must be local" bit set.

Support provided in APAR OW38618 for

- HBB6606 (OS/390 V2R6)
- JBB6607 (OS/390 V2R7)
- HBB6608 (OS/390 V2R8)

Previously, with BPXBATCH, the user needed to set up the environment variable _BPX_SHAREAS.



BPXBATSL is a clone of BPXBATCH. This new version of BPXBATCH will create a local spawn to create the process which runs the requested program.

Why use BPXBATSL over BPXBATCH (w/spawn environment)?

- **If you want a local spawn of their program from BPXBATCH but do not want continued local spawns from their program.**
  - **BPXBATCH:** The _BPX_SHAREAS and other variables needed by BPXBATCH (w/spawn) will be propagated through each environment causing continued local spawns.
    see Fig1.
  - **BPXBATSL:** The environment is **not** set up for local spawns, yet BPXBATSL will force itself to spawn your program. Now all subsequent spawns will be non-local as requested.
    see Fig 2.

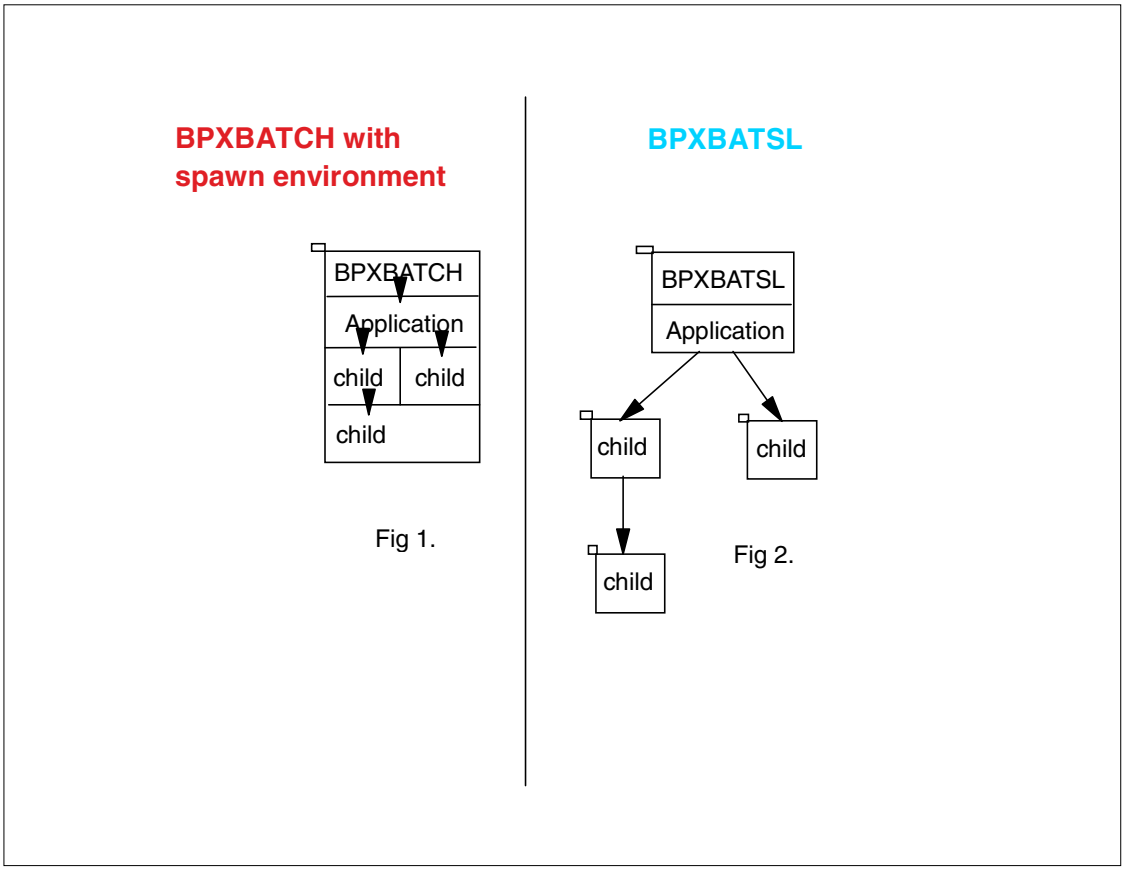


Fig 1. is what the address space configuration looks like when BPXBATCH is invoked and local spawn is requested. As you can see here, the local spawn request is propagated throughout.
Fig 2. with BPXBATSL you get exactly what the user wanted. The main application is spawned locally but not any of the child processes.

pax / tar enhancement

- **pax** and **tar** now support link names greater than 100 bytes
- **pax**:
  - saved in USTAR (default) format archive
  - only restored on OS/390 R9+
- **tar**:
  - saved in USTAR (must be specified by -U) format archive
  - only restored on OS/390 R9+
- When unwound on non-OS/390 platforms, or earlier releases, all files will be restored except the long hard links

pax and tar are utilities used to package a directory tree into a single archive file.

The header records contained within the archive are defined by UNIX standards (for portability between different UNIX platforms). The current definition for a link name (hard link or symbolic link) is max 100 characters.

This is an OS/390 extension, as explicitly allowed in the standards.

Maintains portability with other UNIX platforms. The other platforms will ignore the OS/390 unique headers.

ReIPL Avoidance

- Provide a coordinated method for a product's Kernel resident module to be deleted and reloaded by the product code
- Decreases the need for IPL for some APAR maintenance
- No externals or changes needed by the customer
- TCP/IP uses this feature for their service

The Kernel resident part of these products is called a Physical File System (PFS) and this bridges between the Unix Kernel and a product such as TCP/IP or HFS.
The PFS load module can be deleted by recycling the PFS which is done with two calls to pfsctl(BPX1PCT): PC#RecyclePFS is issued when the product is being stopped. This Quiesces the PFS and when all calls in progress have finished the PFS is terminated. PC#RestartPFS is issued when the product is restarted. This causes the PFS to be reattached and it goes through a normal initialization sequence.

* NETWORK satements from BPXPRMxx are replayed
* MOUNT statements from BPXPRMxx are NOT replayed

A dormant PFS is not easily observable as being any different from when TCP/IP was stopped before. There is only a difference in the Reason_code issued if one attempts to create a new socket, or to use an old socket, after TCP/IP is stopped..

BPXPRMxx Parmlib Member Syntax Checker

- Provide an option on the SETOMVS command to syntax check a BPXPRMxx parmlib member **before** IPL
- Avoids OMVS initialization in minimum-mode for syntax errors

Many customers have complained that it is too late in IPL when they find out if they have a syntax error in their BPXPRMxx member. Message BPXI029I is issued - OMVS= parmlib member not found or is in error.
Error messages are sent to the log which are not available at this point in the IPL. If customer just continues, OMVS comes up in min-mode and a re-ipl is needed to change any of the parms not changeable via the SETOMVS command.

BPXPRMxx syntax checker

SETOMVS SYNTAXCHECK=(xx)

- **Runs the same logic which is used at IPL or via SETOMVS**
- **Any errors cause the same error messages to be written to the syslog.**
- **BPXO039I SETOMVS SYNTAXCHECK COMMAND SUCCESSFUL.**
- **BPXO023I THE PARMLIB MEMBER BPXPRMXX CONTAINS SYNTAX ERRORS. REFER TO HARD COPY LOG FOR MESSAGES.**

Customer complaints about the disasterous effects of a bad BPXPRMxx parmlib member at IPL, especially in a sysplex have triggered the addition of syntax checker support. Simply point the SETOMVS command at a BPXPRMxx parmlib member and it will perform validity checking. This will only catch syntax errors and will not identify problems with FILESYS or MOUNT statements which point to modules or file names which do not exist.

Joblog to stderr

- Improve diagnostics for UNIX user program abends
- Provides the ability to capture WTO messages into an open file
- Messages captured in the file include
  - ABEND symptom dump messages
  - Allocation messages
  - Other system status messages
  - New messages from the kernel for on kill -9 exit status causes and fork/spawn child errors
- Capture only works for forked address spaces (BPXAS)

There have been many instances where an OS/390 UNIX user program fails (abends), the task terminates and the kernel sets up the ending status to say that the process was killed. The end shell user is simply told that the process received a SIGKILL signal which is not true. All MVS diagnostic information that was written to the user JOBLOG is in the bit bucket. These users frequently have to resort to guessing and printfs to figure out what went wrong.

Joblog to stderr

New environment variable _BPXK_JOBLOG

- **_BPXK_JOBLOG=fd**
  - where *fd* is an open file descriptor where messages written
- **_BPXK_JOBLOG=STDERR**
  - Messages written to file descriptor 2, fd 2 must be open for message capture to occur
- **_BPXK_JOBLOG=NONE**
  _BPXK_JOBLOG=
  unset _BPXK_JOBLOG
  - Will terminate message capture
- Environment variable will be propagated on fork and spawn
- Environment variable can be overridden on exec and spawn

```
exec 3>>myjoblog
export _BPXK_JOBLOG=3
```

dbx Enhancements

- **Support debugging with r/w locks and shared mutexes**
- **dbx debugging of programs with long long (64 bit) data types**

dbx will modify the existing mutex objects "$mX" add new information for shared mutexes.
For read/write locks, dbx will create a new object "$lX" to contain the read/write lock information. A new set flag "$lv_events" will turn/off tracing of events to the object. A new command "readwritelock" will display the read/write lock information. The "readwritelock" command will have a new dbx alias "rwl".
The C compiler has added symbolic support for long long primary data types since 2.6 which includes a software simulation of the 64 bit data types. dbx needs to allow users to debug C/C++ programs that include long long and unsigned long long data types.

Migration Actions for V2R9

- File attributes of **mesg**, **talk**, **write**, and **uucp** utilities must be set
  - After defining userid: UUCP
    and groupids: UUCPG, TTY
    - Define with same UID / GID in every security database
- Previously set in /etc/rc
  - Must be removed from /etc/rc to allow root to be mounted read-only
- New sample job: FOMISCHO
  - Invokes FOMISCH1
  - Run during customization

References

- OS/390 V2R9.0 UNIX System Services Planning (SC28-1890-09)
- OS/390 V2R9.0 UNIX System Services Command Reference (SC28-1892-08)
- OS/390 V2R9.0 UNIX System Services User's Guide (SC28-1891-08)
- http://www.ibm.com/s390/unix