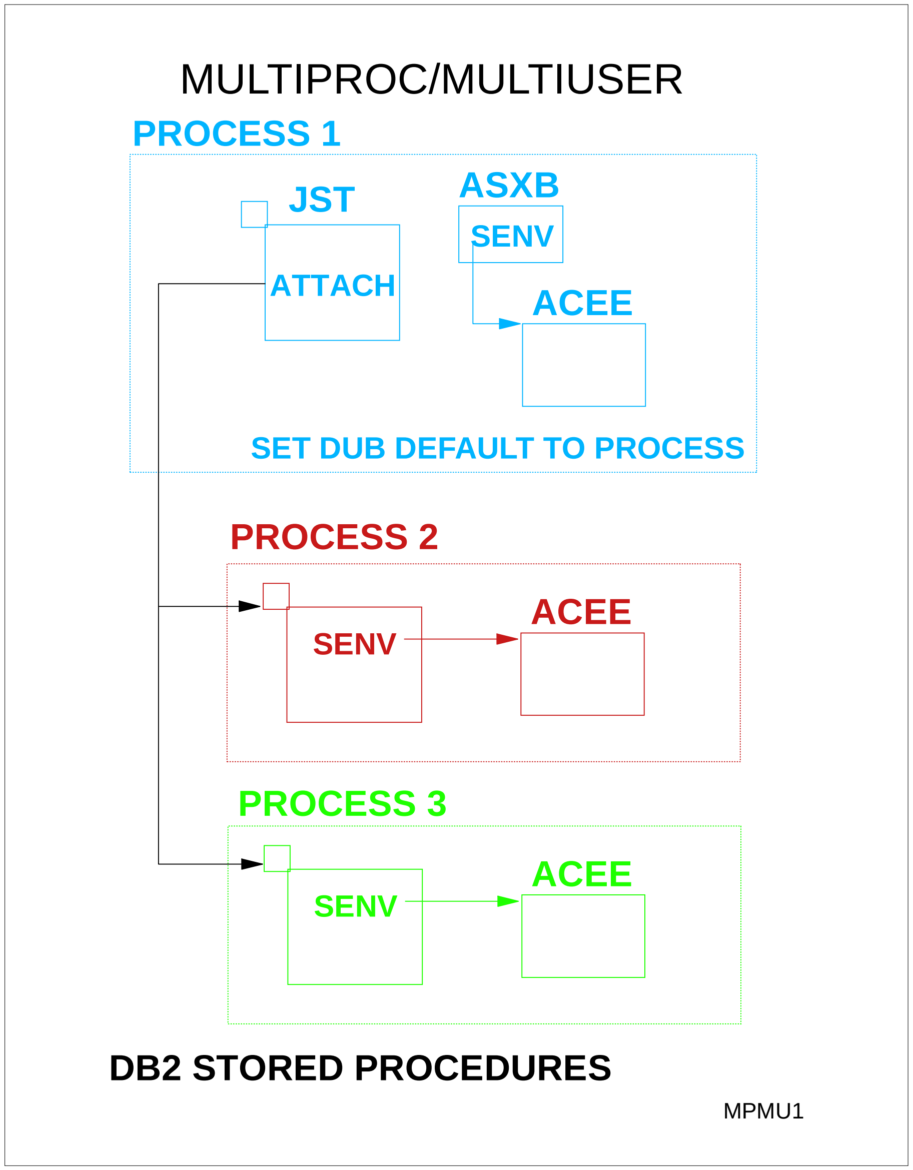
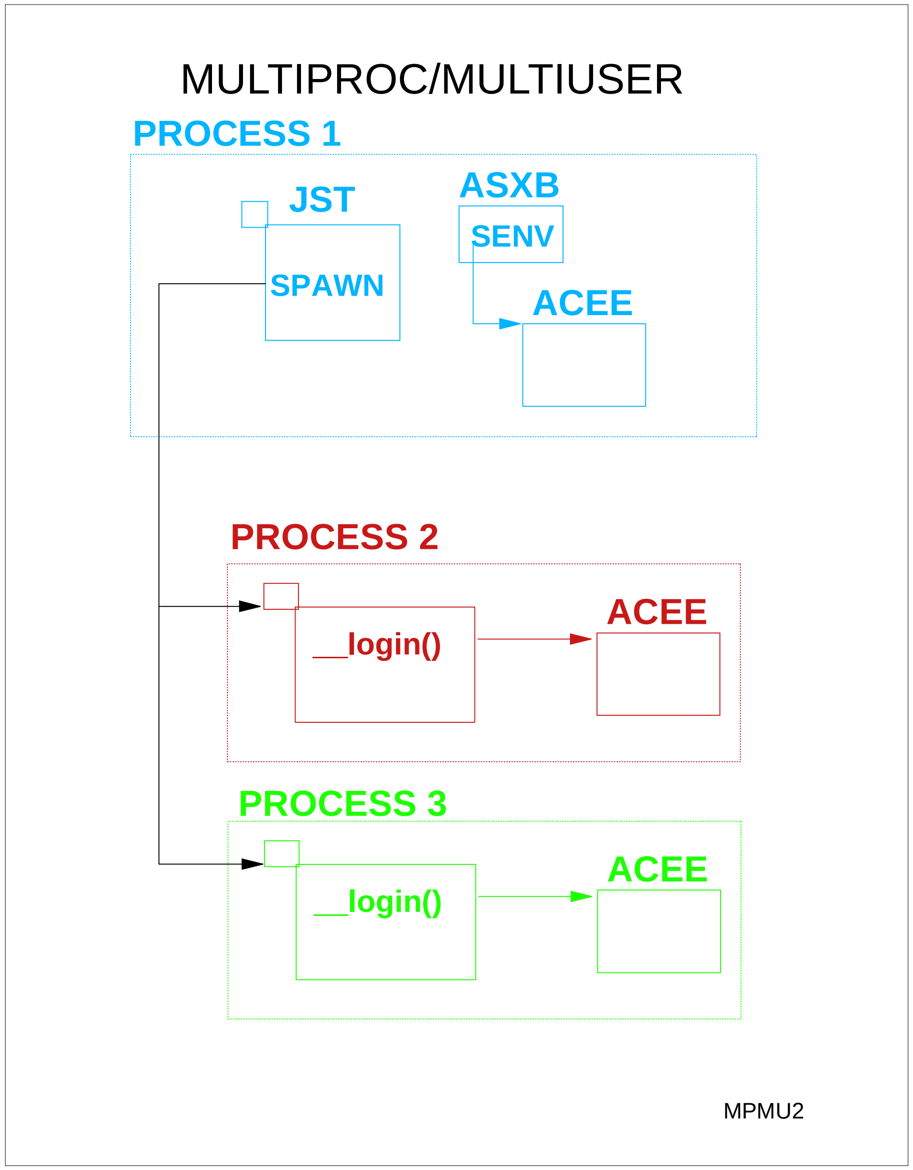
This presentation will cover enhancements in OS/390 UNIX System Services for OS/390 V2R6 and OS/390 V2R7.

OS/390 UNIX SYSTEM SERVICES KERNEL UPDATE FOR OS/390 V2R6 and V2R7

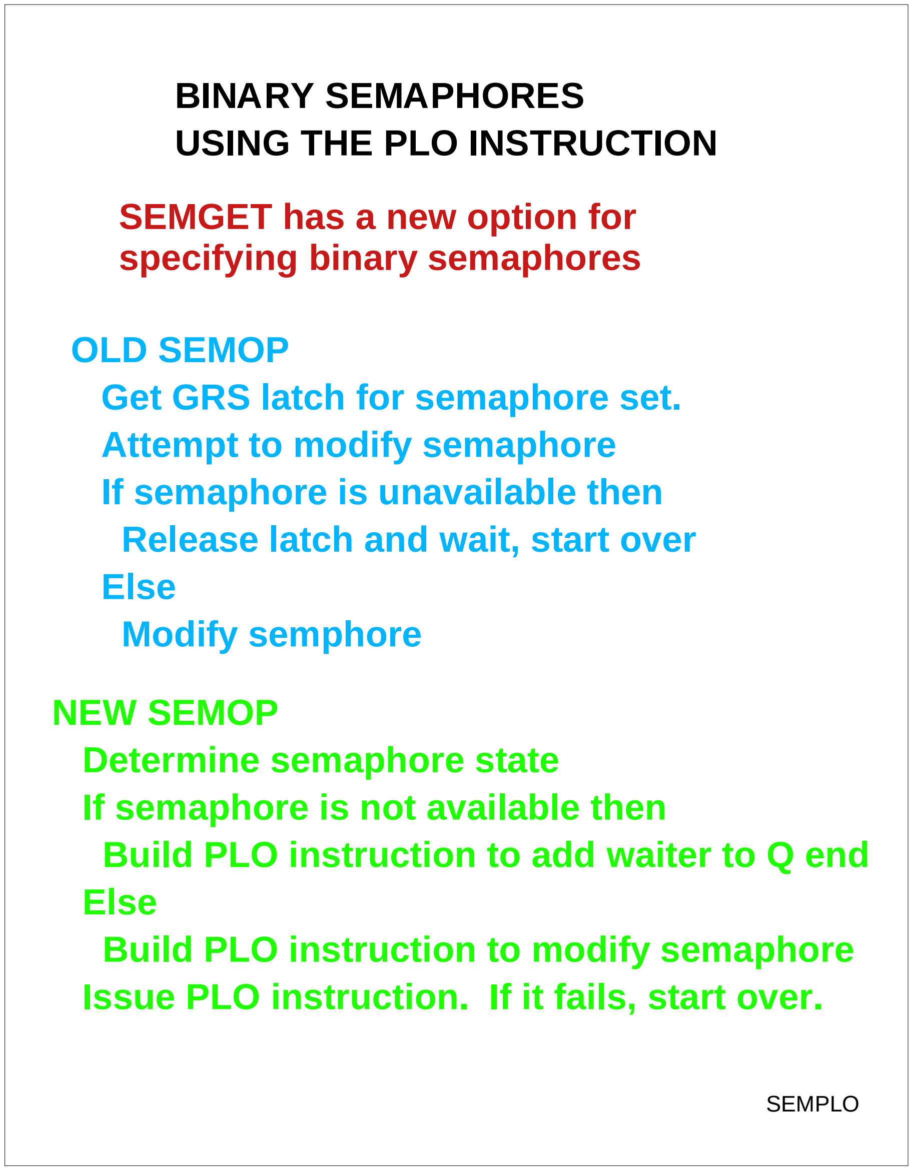
Don Ault
ault@us.ibm.com
May 1998



The support for multiple processes in a single address space with multiple identities is initially supported in OS/390 V2R6. Some of this support was rolled back to V2R4.
This foil shows the case of an APF authorized address space where the main task has called set dub default to tell the kernel that subtasks should be dubbed as processes. The application attaches the subtasks and creates task level ACEEs for the clients that it is doing work for. When the kernel gets control for the first syscall on these tasks, it notices that there is a task level ACEE and dubs the task as a new process, using the user identity to extract the UID/GID and groups that it is to run with.
This support is currently being used by DB2 to run stored procedures which execute UNIX style code.



V2R6: A second phase of multiproc/multiuser is the creation of a new C service called __login. In this scenario, the application is not running APF authorized, but is superuser and permitted to BPX.DAEMON. The server creates a new process via a local spawn for each client. This is made possible by another enhancement to the spawn service to allow the application to request that the child process only be created in the same address space. Prior to this, a local spawn may have been made non-local if the address space was running out of storage. With the new option, the spawn is either local or rejected (EAGAIN).
So once the local process is created, the server verifies the identity of the client and calls __login which causes the kernel to build a task level ACEE for this task and process. This process can call exec and the kernel will preserve the task level ACEE. But, ifthis process does another local spawn, the identity of the child will get the address space identity. It is necessary to do another __login in the child to pick up the right identity.



BINARY SEMAPHORES USING THE PLO INSTRUCTION

SEMGET has a new option for specifying binary semaphores

```
OLD SEMOP
   Get GRS latch for semaphore set.
   Attempt to modify semaphore
   If semaphore is unavailable then
     Release latch and wait, start over
   Else
     Modify semphore

NEW SEMOP
   Determine semaphore state
   If semaphore is not available then
     Build PLO instruction to add waiter to Q end
   Else
     Build PLO instruction to modify semaphore
   Issue PLO instruction.  If it fails, start over.
```

SEMPLO

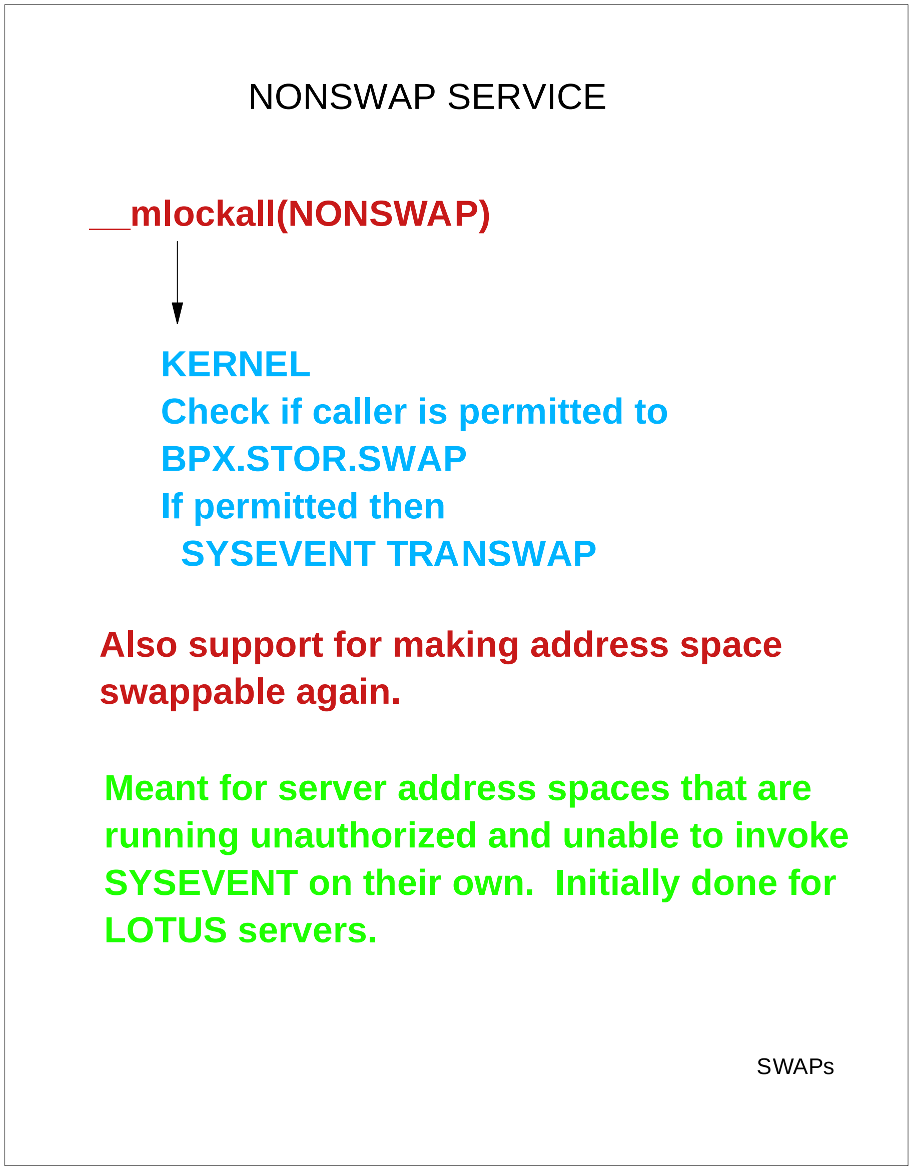
V2R6: OS/390 UNIX has had semaphore support for counting semaphores since MVS/ESA 5.2.2. This support is compliant with all of the standards. The problem is that this support is slow. Since most applications use semaphores as binary semaphores, we have been doing redesign of the semaphore logic to take advantage of the simpler binary semaphore requirements. In other semaphore enhancements, we have simply detected binary semaphore behavior and let this trigger more optimal behavior.

With the introduction of the PLO (Perform Locked Operation) instruction, we require the caller of semget to specify a new BIN_SEM option which indicates that the caller will only be using the semaphore in a binary manner. That is, only semop calls for -1 or +1 and a semaphore value will never be anything but 1 (available) or 0 (not available).

Assuming the PLO hardware is available (CMOS hardware only), the semop processing will be implemented with the PLO instruction. This completely eliminates the GRS latch with significant reduction in pathlength and contention.

If the PLO instruction is not installed, semop processing will use a GRS latch as before.

This support was rolled back to OS/390 R4 via APAR OW32071.



NONSWAP SERVICE

__mlockall(NONSWAP)

$\downarrow$

KERNEL
Check if caller is permitted to
BPX.STOR.SWAP
If permitted then
SYSEVENT TRANSWAP

Also support for making address space swappable again.

Meant for server address spaces that are running unauthorized and unable to invoke SYSEVENT on their own. Initially done for LOTUS servers.

V2R6: The __mlockall service was created to mimic the mlockall() C function defined in UNIX 98. The mlockall service mostly deals with allowing the caller to page fix pages and address spaces. Since there is generally no need for a regular C application to page fix pages, we did the next best thing which is to allow them to become nonswappable. To prevent misuse of this capability, the kernel checks to see if the caller is permitted to the BPX.STOR.SWAP FACILITY class profile. Only if the caller is permitted to this profile will the kernel issue the SYSEVENT TRANSWAP to make the address space nonswapable.
This support was done to improve the performance of LOTUS server spaces, but is available to all applications with the appropriate permission.
A warning to those considering using this. The SYSEVENT TRANSWAP for an address space that will be up permanently will likely restrict the ability of the system to configure storage offline. This is because the LSQA of the target address space was originally allocated in non-preferred storage. When the address space is marked non-swappable, there is no way for the operating system to move the LSQA to preferred storage.
This support was rolled back to V2R4.

MISCELLANEOUS

The __console service has been enhanced to allow the caller to issue multiline WTOs instead of the previous single line messages.

A set of C functions and a kernel service have been provided to give C programs access to most of the WLM services. Callers must be permitted to BPX.WLMSERVER.

The ability to DBX attach to an APF authorized address space has been added. The caller must be permitted to BPX.DEBUG.

The ability to classify UNIX address spaces by jobname has been added to WLM. OW30439 has this support.

V2R6:
The __console service provides the ability for an unauthorized C program to respond to STOP and MODIFY commands. In addition, it allows the C application to write to the console via WTO. This new support extends that ability by allowing the caller to write multiline WTOs.

C functions for WLM: CheckSchEnv(), ConnectServer(), ConnectWorkMgr( ), ContinueWorkUnit(), CreateWorkUnit(), DeleteWorkUnit( ), __DisconnectServer( ), JoinWorkUnit(), QueryMetrics(), QuerySchEnv(), LeaveWorkUnit( ), and more.

In the past, there was no way for a DBX debugger to debug an APF authorized server. If the debugger is permitted to BPX.DEBUG facility class profile, the kernel will now allow DBX to debug the APF authorized process.

In prior release, the only way to tell WLM to distinguish between work running in forked address spaces was by USERID or accounting information. Since many daemons are set up to run with the same userid and have no accounting data, it was difficult to have WLM treat certain daemons differently. Now WLM can be set up to classify a forked address space by the jobname. The shell allows you to export _BPX_JOBNAME to set a desired jobname and now these server spaces can be correctly classified.

MISCELLANEOUS

During fork and spawn, the initial REGION size is set to 50M instead of 0 to prevent IEFUSI exits from getting involved. OW32459

F BPXOINIT,SHUTDOWN=FORKINIT Provides a mechanism to clean up WLM fork initiators prior to a shutdown or P JES.

_BPX_SHAREAS=MUST allows the application to request a local spawn or no spawn at all.

OW30182 changes time out behavior so that TSO users will terminate when JWT is detected. Terminating signals now detect TGET and TPUT waits and break through.

Fork, pthread_create and ptrace support for IEEE Floating Point registers.

V2R6: During fork and spawn, the IEFUSI exit used to see a region size of zero. Some exits were setting a new region size which was interfering with fork. Now the region size is set to 50M to sooth the IEFUSI exit. Region size is propagated from the parent during fork child processing.
The F BPXOINIT,SHUTDOWN=FORKINIT operator command was added after complaints about waiting 30 minutes for the initiators to time out following a P JES2 command.
New setting of _BPX_SHAREAS=MUST allows the application to force a local spawn. Useful when the first process has DDs needed by the second process.
The kernel propagates the new floating point regs on fork. On pthread_create, floating point controls are propagated to the new thread. Ptrace provides the hooks for DBX to access new FP regs.

SERVICEABILITY

**CTRACE COMP(SYSOMVS)
OPTIONS((SCCOUNTS))
provides syscall counts usefull for primitive profiling of your C applications.**

**CTRACE COMP(SYSOMVS)
OPTIONS((SEARCH(off,len,string)))
provides the ability to only print trace entries which match the search argument.**

CTRACE records for open, exec and spawn now have the filename included.

Syscall failures are now always traced.

CTRACE buffer size is overridden to the 4M max when options specified with default buffer size.

OMVSDATA NETSTAT

V2R6: A SCCOUNTS (SysCall COUNTS) option was added to the SYSOMVS ctrace. When requested, this will print out a report showing total syscalls and frequency. This information can be used against any trace data, but is meant for running against a ctrace run to an external writter. This will allow an application writter to understand the frequency of syscalls being made.
A new SEARCH argument was added to the SYSOMVS CTRACE IPCS formatting options. This will allow the trace entries to be filtered by TCB, failing syscalls or any other string in the trace entries.
File names were added to open, exec and spawn so you can tell more about what is happening.
Whenever a syscall failure is detected, the results are always traced.
The trace buffer size is upgraded to 4M when options are specified with the minimum buffer size. This is to deal with frequent failures to increase the buffer size.
OMVSDATA NETSTAT formats TCP/IP stack information.

PERFORMANCE

pthread_cond_wait and pthread_cond_post have been redesigned to eliminate some of the kernel calls, thus improving performance.

More kernel data made accessible to RTL.

RACF UID/GID lookup redesigned.

LOADHFS optimized for Job Pack Queue search to assist on frequently accessed DLLs.

Kernel timer queue optimized for dual ended search.

V2R6: Prior support for pthread_cond_wait required the C RTL to call the kernel for pthread_cond_setup, then pthread_cond_wait. The reason for the RTL to call the kernel at all was to enable a signal to wake up the pthread_cond_wait. The kernel has added new ECBs and flags to the Thread Level Information (THLI) block. The RTL uses these new fields to inform the kernel when it is going into a signal enabled wait. The kernel will post an ECB in the THLI when a signal arrives. This was done for performance reasons to cut the pathlength.
pthread_cond_timed_wait builds on the above support, but also provides a fast kernel service for establishing a timer. This timer is automatically cancelled on the next kernel call. This eliminates the 2 STIMER calls previously required.
The kernel has added fields to the BPXYPRLI to allow the RTL to access UIDs, GIDs, parent PID and process group ID.
RACF has redesigned UID/GID lookup to deal with undefined UIDs and GIDs. Dramatic improvements on initial lookup and ls commands against NFS or DFS mounted file systems.
When loading a DLL or locale from the HFS, the algorithm has been optimized to recognize previously loaded modules on the Job Pack Queue.
Kernel timer has been optimized to deal with environments containing over 1000 elements.

FILE SYSTEM BUNDLED SYSCALLS

__open_stat() - BPX2OPN

Opens the file and returns the data as if an fstat were done after the open.

__accept_and_recv() - BPX1ANR

Combines the first recv with the accept and only returns to the caller when the data from the accept is in the buffer.

send_file() - BPX1SF

Allows a file to be sent out a socket with a single kernel call. Caller can provide header and trailer information.

V2R6: Some common pairs of file system calls have been combined to reduce the pathlength.
Open followed by stat is commonly used and can now be replaced with a call to __open_stat(). This service has a new stat area parmameter in addition to the parms already supported on open.
The web frequently does accept followed by receive when a new connection arrives. The __accept_and_recv() function speeds this up by only returning to the caller when the data for the new connection has arrived.
The send_file() function was added to assist the web and ftp type servers that need to send an entire file to another destination. This call currently eliminates the multiple kernel calls required to fill buffers from the HFS and then the send calls. In the future, internal optimization of buffers between the HFS and TCP/IP is anticipated.

OTHER FILE SYSTEM UPDATES

srx_np(BPX1SRX)

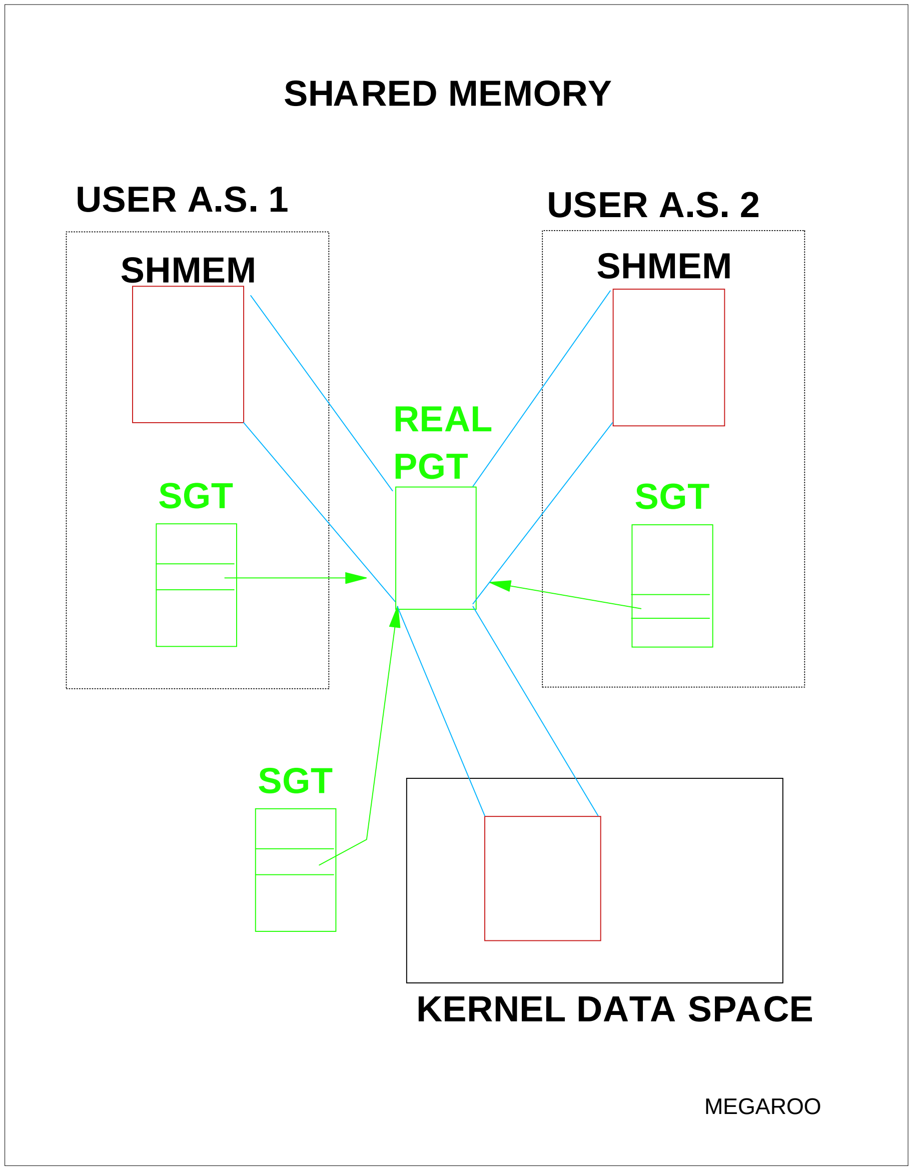
TCP/IP send and receive with buffer ownership passing (key 6 buffers).

Receive_with_timer

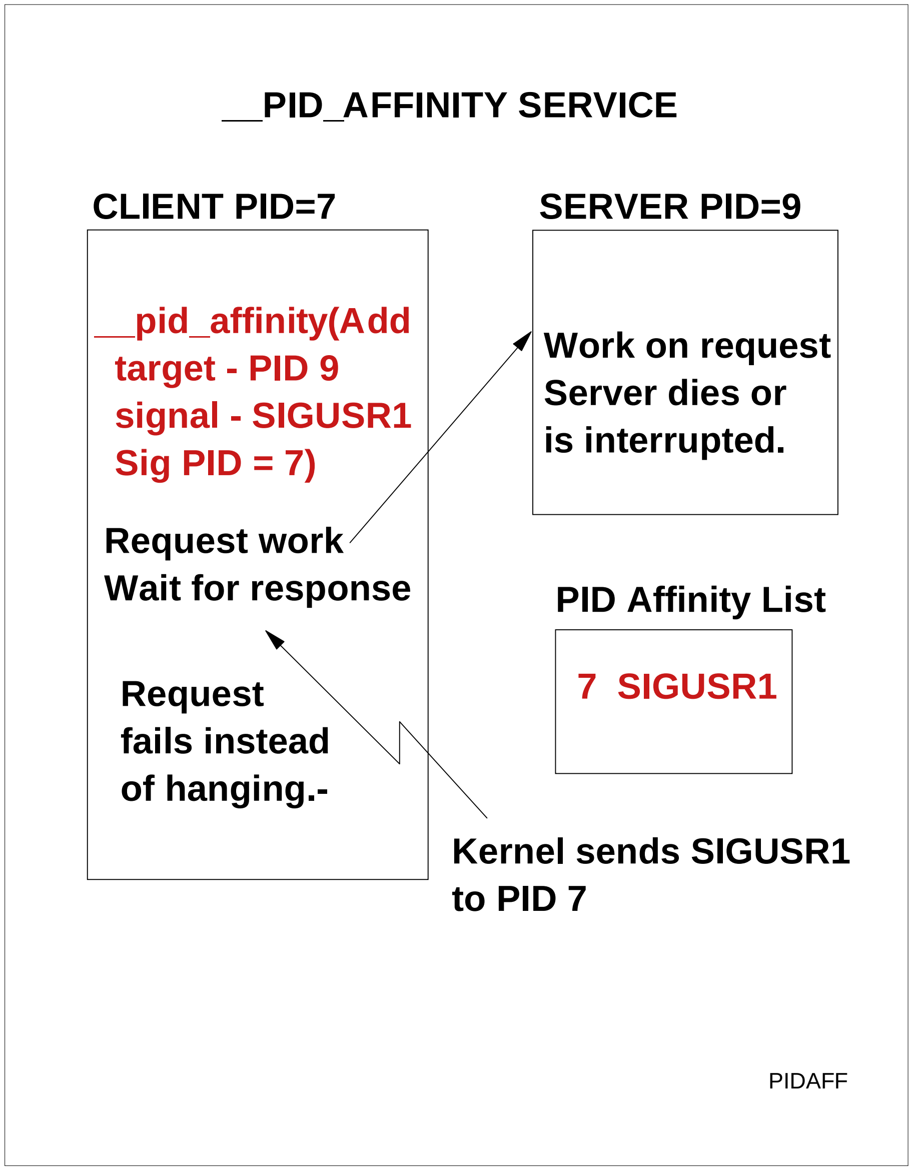
Can be achieved with AIO support. Replace select which is frequently used just to get receive with timeout.

Support in Common Inet for subnet and supernet masks.

V2 R6:
The srx_np service was primarily created for FTP. It allows the caller to obtain buffers from the CSM buffer service. These buffers can be filled on a read() call and then given directly to TCP/IP on a send(), thus eliminating a data copy.
Only glitch is caller must be authorized and running in key 6.
In the past, when an application wanted to wait on a socket with a timeout, they had to use select(). Now, by using the functions in Asynchronous I/O (AIO), you can perform a receive with a timeout and not undergo the overhead of select.



V2R6: With the previous version of shared memory, each page of shared memory used a 32 byte RSM control block in ESQA and each address space had its own page tables. If you tried to share 500 Meg across 500 address spaces, it would have consumed 2 GIG of ESQA for the control blocks and 1 GIG of LSQA for the page tables.
With the new support, the caller of shmget can request megabyte level sharing. When this is done, the kernel calls RSM to set up shared page
tables. Now for 500 Meg shared by 500 address spaces, it will only consume 2 Meg for 1 set of page tables.
With this new option, the application must agree to have all sharing address spaces maintain the same view of the storage.



V2R6: The PID affinity service (C function __pid_affinity) was created to allow some level of recovery for client/server connections or server to server connections. When you have 2 processes communicating via message queues or shared memory and one of those processes terminate unexpectedly, it frequently leaves the other process waiting for a response forever.
The PID affinity service allows a process to build a temporary affinity to another process such that termination of that process will result in getting the requested signal sent back to the process. The PID affinity service can be called from the client or the server end. The caller specifies which signal is to be delivered. There is usually a signal catcher set up to catch these signals.

BPXBATCH ENHANCMENTS - R6

```
//UNIXWDDS JOB
//STEP1         EXEC PGM=BPXBATCH
//STDIN     DD PATH='/u/user/input',...
//STDOUT DD PATH='/u/user/output',...
//OTHERDD DD DSN=USED.IN.PGM
//STDENV  DD *
_BPX_BATCH_SPAWN=YES
_BPX_BATCH_UMASK=0755
_BPX_SHAREAS=YES
/*
```

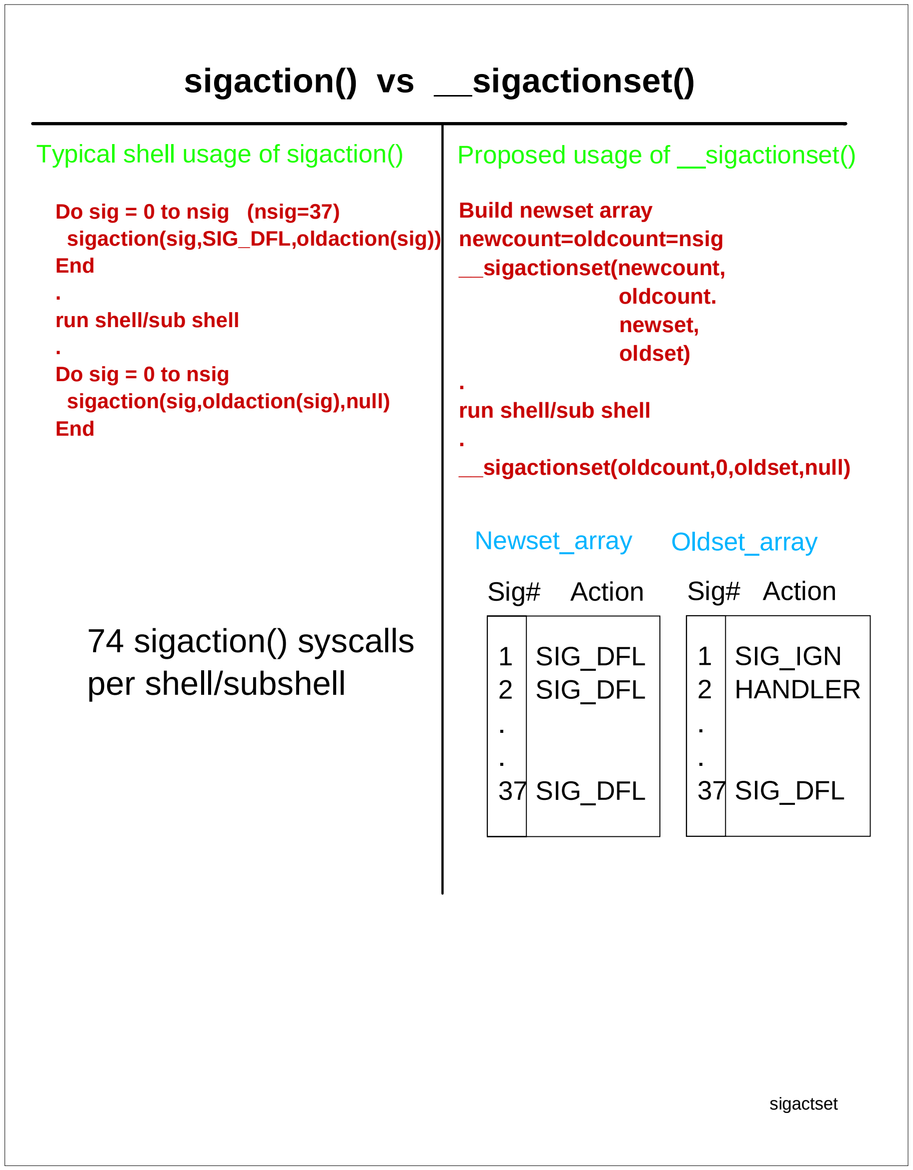
1. **Parm string increased from 100 bytes to 500 bytes. Usable in all but JCL.**
2. **Ability to request spawn and local spawn makes it possible to pass MVS DDs to a UNIX program.**
3. **Support for setting the umask.**

BATCH2

V2R6: A number of enhancements have been made to BPXBATCH. The parm string can now be up to 500 characters. JCL still has the limitation of 100 characters, but BPXBATCH can be called from TSO, REXX or other programs and supports the longer parm string.

With _BPX_BATCH_SPAWN=YES, it is possible to request that BPXBATCH run the requested program via spawn. In conjunction with existing environment variable _BPX_SHAREAS=YES, the spawned program will run in the same address space in a subtask (local process). That means other DDs in the JCL are available to the invoked program, in addition to stdin, stdout and stderr.

With _BPX_BATCH_UMASK, the caller has the ability to set the umask for the program. Since there is no login shell script run for BPXBATCH, this provides one of the capabilities usually satisfied by login scripts.



sigaction() vs __sigactionset()

Typical shell usage of sigaction()

```
Do sig = 0 to nsig   (nsig=37)
  sigaction(sig,SIG_DFL,oldaction(sig))
End
.
run shell/sub shell
.
Do sig = 0 to nsig
  sigaction(sig,oldaction(sig),null)
End
```

74 sigaction() syscalls per shell/subshell

Proposed usage of __sigactionset()

```
Build newset array
newcount=oldcount=nsig
__sigactionset(newcount,
               oldcount.
               newset,
               oldset)
.
run shell/sub shell
.
__sigactionset(oldcount,0,oldset,null)
```

Newset_array

Sig#	Action
1	SIG_DFL
2	SIG_DFL
.	
.	
37	SIG_DFL

Oldset_array

Sig#	Action
1	SIG_IGN
2	HANDLER
.	
.	
37	SIG_DFL

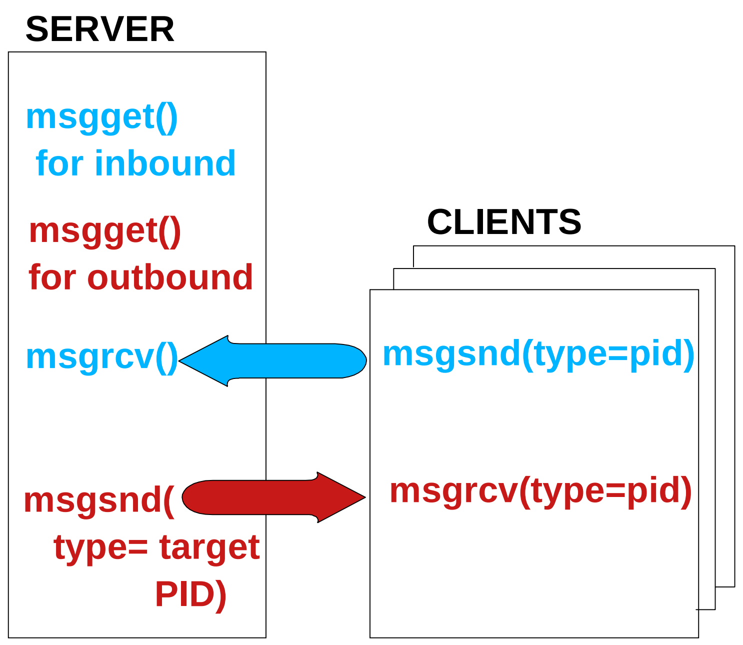
sigactset

Prior to R6, any application wanting to reset signal actions needed to call the sigaction() service for each and every signal that needed to have its action changed. In the scenario where OMVS shell is invoked it is required to run with all signals set to their DEFAULT action. To accomplish this the shell code loops through all possible signals setting the action to DEFAULT and saving the old action. The shell code will then run the shell or subshell cmd. Once that is done the shell code must restore the original signal environment by once again looping through all the signals, resetting the saved action.

With the introduction of __sigactionset() it is now possible for a application to set the entire process signal action environment with one call. The __sigactionset() service takes as input an array of signals and there corresponding actions. The user can also specify an output array that __sigactionset() will use to place the current signal settings into. This output array is of the same format as the input array such that it can be used to restore the signal environment when needed.

By compressing the sigaction() service into the __sigactionset() we can save many syscalls and latch obtains. Also the application does not need to know the implementation specific signals nor how many there are. __sigactionset() allows the user to just fill out an array with 0-63 signals and set them to any action they desire. Even though their may be several signals not defined or several signals that can not have their action changed, __sigactionset() has the ability to ignore undefined signals or invalid actions setting.

PID SPECIFIC MESSAGE QUEUES - R7



1. Clients can only send messages with type=pid
2. Which allows the server to always respond to specific PID on second message queue.
3. Client can only receive message with type equal to their PID.

MSGQPIDS

V2R7: One model of client/server communication is for the server to define 2 message queues where one is for inbound communication and the other is for sending responses to the clients. If the client PID is used as the message type for both directions, it allows the server to know who sent a message and to direct the response to the correct process. For this to work, the server must define the first queue such that any process can write to it and the second queue such that any process can read from it. This opens the server up to possible abuse by the clients.
With this support, clients can only send messages with their PID as the message type and can only receive messages with type equal to their PID. This prevents a process from trying to trick the server into believing it is another process. It also prevents false clients from stealing responses meant for other processes.
Two new flags are defined on msgget to request this behavior.

PERFORMANCE - R7

Optimized security checking in performance sensitive syscalls:

- Message queues
- Semaphores
- File system (stat, open, other lookups)
- ps command

FACILITY CLASS PROFILE:
BPX.SAFFASTPATH
ENABLES THE SUPPORT

DBX storage access optimized to prevent expected abends.

PERFR7

V2R7: All of the security for the OS/390 UNIX functions is implemented through calls to SAF which are routed to the security product. Although these calls have been optimized for pathlength, they can still consume a large portion of the pathlength of certain syscalls. The kernel has implemented an inline macro which performs most of the standard security tests. If these tests pass, the call to SAF is bypassed. Since bypassing SAF eliminates the ability to audit successful calls, the customer must activate this support by defining the FACILITY class profile BPX.SAFFASTPATH. This support eliminates about 500 instructions during file name lookup for each directory in the file name.
Similar savings are achieved in the IPC flows for message queues and semaphores. The PS command also uses this macro.

The DBX calls to PTRACE were triggering multiple expected abends when connecting to modules in storage. The PTRACE logic has been changed to anticipate expected program checks with storage key validation to avoid the abend and the trip through RTM.

RAS STUFF - R7

Error code text

pfsctl() allows applications to retrieve text

PC#ERRORTEXT

REXX exec

bpxmtext reason_code

STARTUP DIAGNOSTICS catches delays for: CAS, RACF, JES, File system init, fork inits, and hangs in /etc/rc processing.

ANALYZE exit for semaphores

F BPXOINIT,SHUTDOWN=FORKS

ps support for threads

RASR7

V2R7: For a long time, folks have been putting the kernel reason codes on the forums and we have been running an internal exec to lookup the text for the reason code.
We have finally shipped this support and made it accessible through multiple formats:
- pfsctl() with a function code of PC#ERRORTEXT will retrieve the text from a programming environment.
- REXX exec BPXMTEXT will accept the reason code and return the error text.
- The env variable _EDC_ADD_ERRNO2=1 will cause the hex reason code to print. Future support will add reason text.

During IPL, the OMVS kernel can run into multiple snags which can prevent the UNIX services from becoming fully functional. Frequently this goes undetected until some later point in time where applications hang or fail due to the OMVS kernel not completing initialization. This support adds code to record the stage of initialization and watch for lack of progress. If a delay in initialization is detected, an operator message is written indicating the phase of startup that has been reached.

An IPCS ANALYZE exit has been written to add contention information to the existing ENQ, latch and lock information already provided by ANALYZE.

There is a new kernel service getthent which is called by a modified ps command. This new support allows the caller to display information for the threads in a multithreaded process. Additional process information is also available.

USER DUMP SUPPORT - R7

New signal SIGDUMP dumps but does not terminate the process.

```
TERMTHDACT(UADUMP)
          (UAONLY)
          (UAIMM)
```

Environment Variable

```
_BPXK_MDUMP=MVS.DATA.SET.NAME
            hfs/file/name
            OFF  (default)
```

```
F BPXOINIT,DUMP=pid
                pid.tid
```

can trigger a dump of the target process.

USERDUMP

V2R7: In the past, if the user could not debug their application with a CEEDUMP or DBX, they had to enlist the support of the system operator to take an SDUMP via the DUMP or SLIP commands.

This support attempts to provide greater user flexibility in getting a dump of their address space.

A new signal SIGDUMP has been created. When a SIGDUMP signal is delivered, it causes an ABEND and RETRY to occur such that ABDUMP is invoked to take a dump, but the process is not terminated (similar to DUMP command).

LE has added new run time options to allow you to control how the dump occurs:

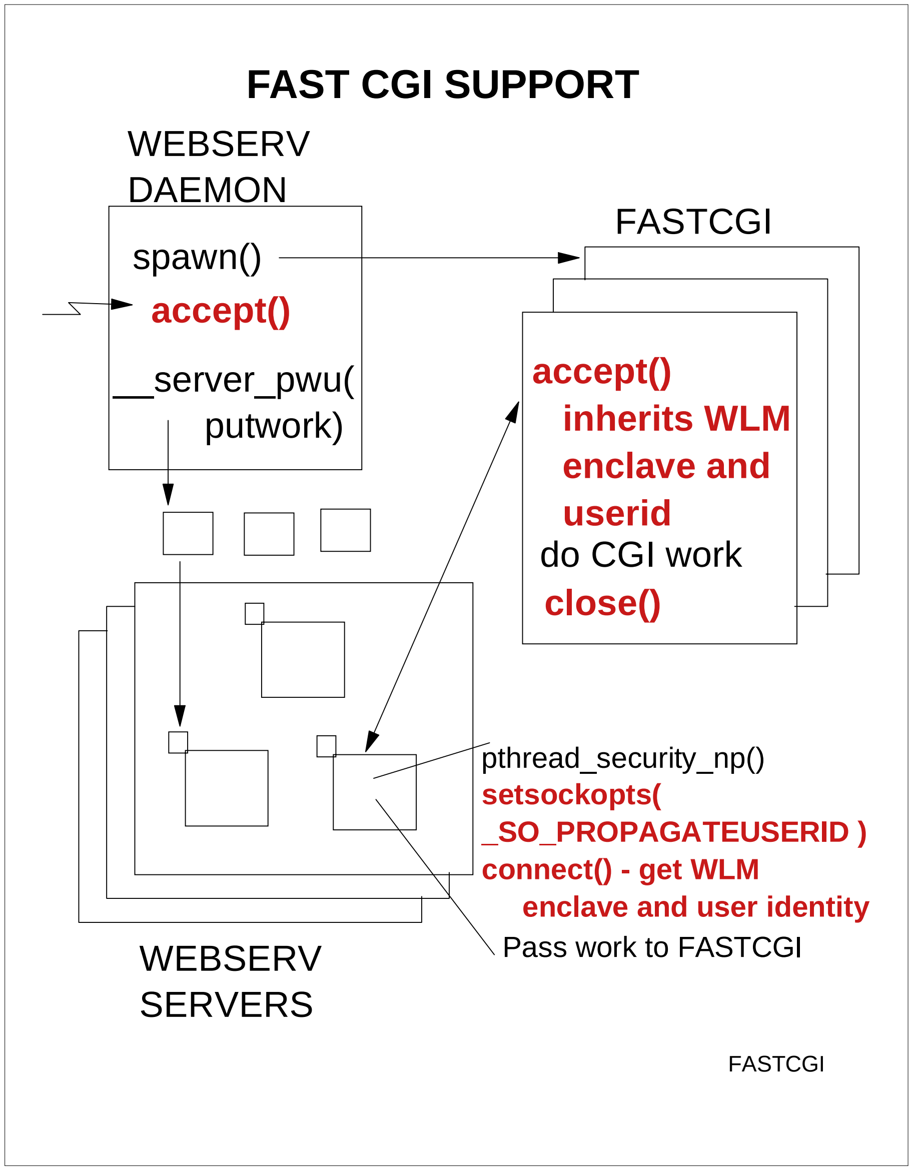
- UADUMP (existing option) allows a system dump in addition to a CEEDUMP.
- UAONLY will just attempt to take a system dump.
- UAIMM will attempt to take a system dump directly from LE's ESTAE.

By system dump, I mean an ABDUMP which will be either a SYSUDUMP, SYSABEND or SYSMDUMP.

A new environment variable _BPXK_MDUMP allows you to specify whether you want the dump taken to an HFS file, an MVS data set or to be suppressed. This support will only kick in if running in an address space with no ABDUMP DDs. It is meant mainly as a means to get a dump in a forked or spawned address space where DDs are hard to come by.

The dump can be triggered in multiple ways:

- The kill command from the shell supports the SIGDUMP signal.
- The F BPXOINIT,DUMP command allows you to target the dump to a particular process or thread.
- There are signals in UNIX that traditionally trigger dumps to occur. These signals will now trigger dumps based on the LE dump options above.



V2R6: Support has been added to enable the WEB server to run %%CLIENT%% CGI programs in a faster manner. The current implementation requires the WEB server threads to issue a spawn to pass control to the CGI program. The overhead of spawn is significant.

With FASTCGI, the web first creates a pool of FASTCGI address spaces using spawn. These address spaces open an AF/UNIX socket and go into an accept() to wait for a CGI transaction. This accept is recognized as special by the accept function for AF/UNIX sockets.

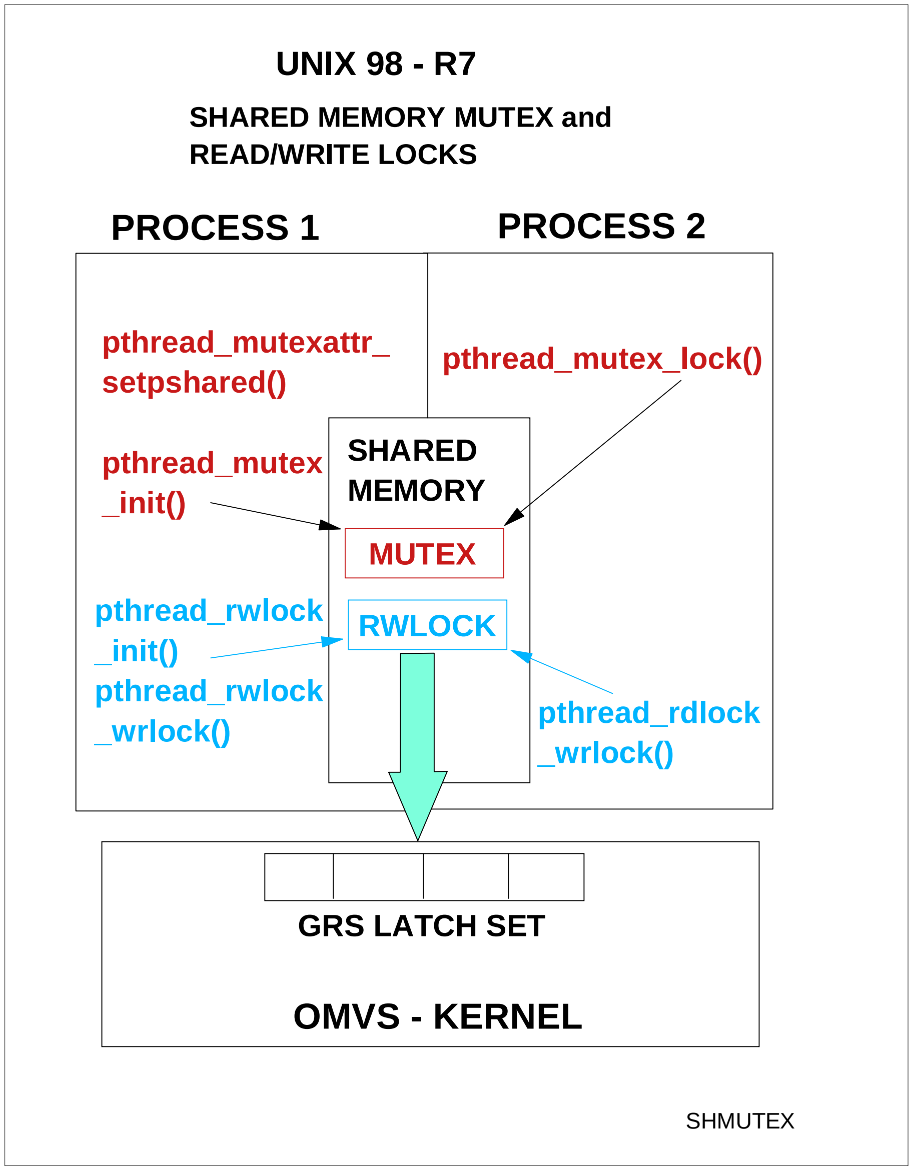
As CGI requests come into the web daemon, the socket connection is accepted and passed to a web server thread via __server_pwu. This creates a WLM enclave representing this request. The web server thread does a __server_pwu call to get the work and inherit the WLM enclave.

The web server thread then does a setsockopt to inform the kernel that this is a special socket.

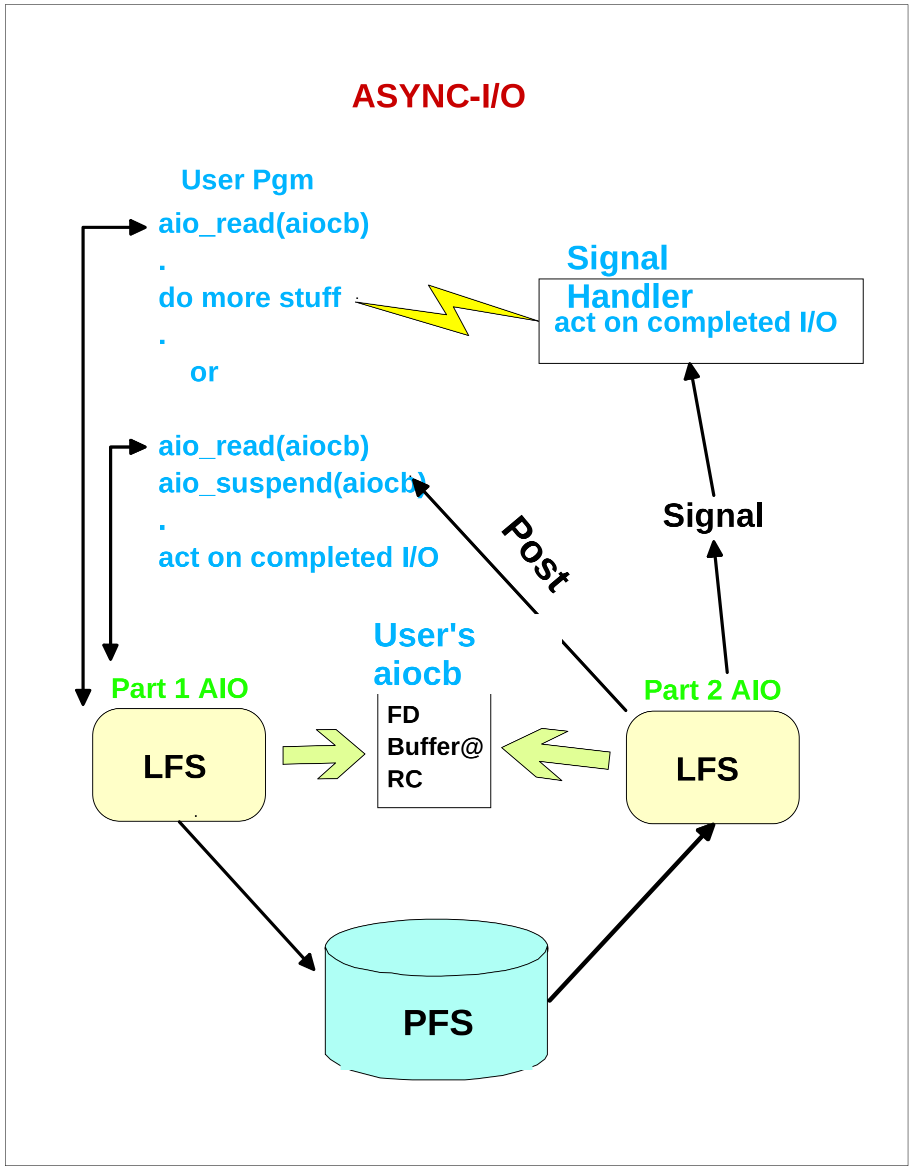
On the connect call, the kernel takes the user identity and WLM enclave token and passes it through to the accept syscall which is in a wait. As part of the accept flow, the kernel changes the idenity of the FASTCGI address space to match that of the thread which issued pthread-security_np. The WLM enclave is also propagated to the FASTCGI space. The WLM enclave and user identity are cleaned up when the socket is closed.

DBX ENHANCEMENTS - R7

Performance - compiled opt(2)
New command: onload
New support to deal with ASCII
Better handling of DLLs
dbx configuration/setup file:.dbxsetup
Improved symbol processing
detection and reporting of bad compiler symbolics
Support __cdecl in compiler's demangled names.
Allow correct debugging of single line functions.
Allow debuggee program arguments on the dbx invocation
echo to user ptrace() call/return info on errors
Extensive support for IEEE Floating Point regs.



V2R7: The following new functions have been created to support shared mutexes and shared read/write locks defined in the UNIX 98 standard:
pthread_mutexattr_getpshared()
pthread_mutexattr_setpshared()
pthread_rwlock_destroy()
pthread_rwlock_init()
pthread_rwlock_rdlock()
pthread_rwlock_tryrdlock()
pthread_rwlock_trywrlock()
pthread_rwlock_unlock()
pthread_rwlock_wrlock()
pthread_rwlockattr_destroy()
pthread_rwlockattr_getpshared()
pthread_rwlockattr_init()
pthread_rwlockattr_setpshared()
Existing mutex functions changed to recognize and support mutexes which reside in shared memory. Locking functions can be used instead of semaphores.
Locks are mapped to GRS latches inside the kernel. This enables D GRS,C contention analysis and IPCS ANALYZE support for free.



With async I/O applications may do one or more reads or writes to a file or socket and have control returned without having to wait for the I/O to complete. Several Async I/O functions have been implemented on R7:

aio_read() - read file/socket data to specified buffer
aio_write() - write from specified buffer to file/socket
aio_suspend() - wait on list of aiocbs for aio to complete
aio_cancel() - cancel outstanding aio request
aio_error() - get error status aio operation
aio_return() - get return info of completed aio operation

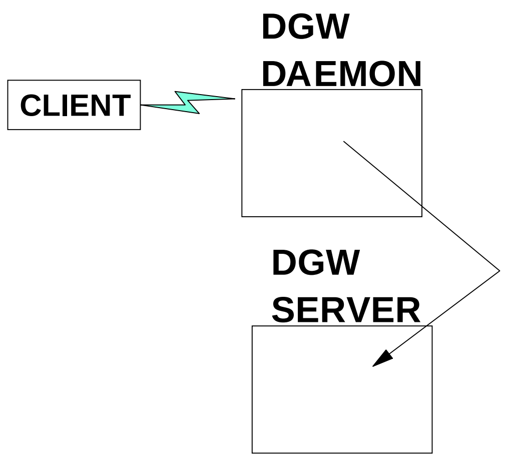
These functions may be used by an application to kick off several I/O requests where each request is represented by an aiocb. The aiocb is a user control data area that defines the async I/O request and contains such information as the File Descriptor, buffer@, notification event type and the return code of the request.

The notification event type may be of several flavors. The application may opt for no notification, notification by running an exit, posting of an ecb or having a signal generated to the process.

In this foil we see two types of flows, one where a user issues a async I/O request(with signal notification), gets control back and continues running while the I/O is being handled by the PFS. When the I/O completes the LFS again gains control, reads the requested information into the buffer pointed to by the aiocb, sets the return code of the read and generates the user requested signal to the process. The signal interrupts the user and control is given to the user's signal handler. From here the I/O completion can be acted upon by the application.

In the other scenario the application issues an async I/O request and then waits in the aio_suspend function. In this case no notification was spec

SSL CERTIFICATE SUPPORT



- Web client passes SLL certificate.
- If certificate is not already registered
- Prompt for userid/password
- Call __passwd service to authenticate
- __certificate(certificate, REGISTER)

CERTREG

V2R6: Support has been added to the system to allow a web client to register a certificate for the calling client. This eliminates the step where the client must send their certificate to the host and involve the system administrator to run a RACF certificate upload job.
Now the client connects to the web site. The customer must define a URL which will accept a SSL certificate. If the certificate is not already defined to the security product, then the web code prompts the client for their userid and password. The client identity is then established on the thread using pthread_security_np. Then the new __certificate() service is called to register the new certificate. The certificate is registered to the user that is currently defined in the task level ACEE.
Support is also provided in __certificate() to deregister a SSL certificate.

SHELL & UTILITIES - R7

Support link and unlink shell commands.
UNIX 98

pax external link support

Recompiled with OPT(2) and IPA for improved performance.

Performance tune up for:
vi find pax tar cpio compress uncompress zcat grep ar ls du

ps for thread level information and additional process information.

V2R7:
link - create a hard link to a file - same as ln
unlink - remove a directory entry - same as rm
chroot - change the root directory for the execution of a command
pax/tar - allow it to store and extract external link files and preserves the extended attributes (APF, PROGCTL, NOSHAREAS) associated with an HFS file.
The ps utility has new thread-specific output fields. (e.g. TID and TAGDATA) New thread-related fields for processes will also be created such as thdcnt, which displays the total number of threads associated with a process.

INSTALLATION IMPROVEMENTS

SMP/E R7

- Support for Shell Scripts
- Support for symbolic links

Filesystem - R7

- Dynamic creation of character special files (/dev/*)
- Non-secure filesystems
- Upon Dub failure, use root for HOME directory

Utilities - R7

- BPXCOPY support of symbolic links and extattr bits
- SETEUID support in:

\ TSO MOUNT and UNMOUNT commands
SMP/E
BINDER
BPXCOPY

INSTALL1

V2R7: SMP/E has been enhanced to allow the install logic to execute a shell script following other installation steps. This eliminates the need to run post install jobs to touch up the file system. SMP/E can now define symbolic links.
Based on demand, the file system now dynamically creates the /dev/ character special files, eliminating the post install step.
MOUNT supports an option to indicate that the mounted file system is not secure. The system will not honor setuid, setgid, APF or program control attributes of a non-secure file system.
During DUB, lack of a home directory will trigger use of the ROOT for HOME. Allows ROOT to be unmounted.
BPXCOPY supports symbolic links, extended attributes (APF, PROGCTL, NOSHAREAS), set owner UID/GID.
MOUNT, UNMOUNT, BPXCOPY, the binder and SMP/E will now perform a seteuid to 0 if not running under uid 0. Requires BPX.SUPERUSER permission.

INSTALLATION IMPROVEMENTS

Software Manufacturing - R6
- **Support for Single HFS Delivery**

Binder
- **Support for symbolic links and extattr bits**

Packaging Standards
- **Create packaging standards for all products installing in the HFS to follow**
- **Provide samples, naming convention, packaging techniques etc.**

Documentation Improvements
- **OS/390 UNIX Planning**
- **Planning for Installation**
- **ServerPac - In your order**
- **Program Directory**

INSTALL2

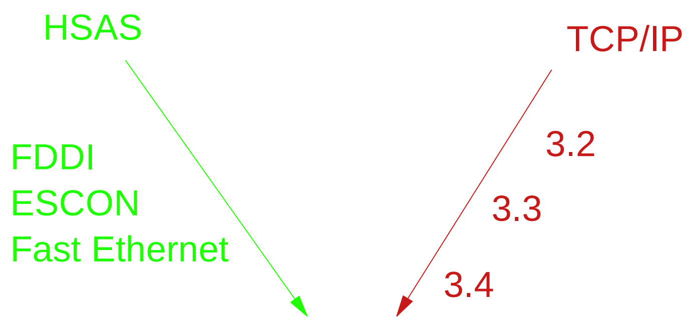
V2R6: OS/390 now comes with one large HFS instead of many smaller HFS's for each component. Customers may want to split it up later.
The Binder has added support to set symbolic links and the APF and program control external attribute flags.
Standards will be documented for packaging products with parts in the HFS.
Additional documentation is being provided to assist with systems managment tasks involving OS/390 UNIX installation and customization.

HIGH SPEED ACCESS SERVICES

GC31-8676 - Covers configuration and use

Webstones on 10/way 4600 conn/sec
at 80% CPU ran out of clients
AFPA prototype doubles thruput.

130 Meg/sec outbound - 1500 MTU
175 Meg/sec inbound
OS/390 to OS/390 with ESCON.



HSAS sets the bar for TCP/IP. When they coverge, we will have 1 stack.

V2R6: Yes we still have 2 stacks on OS/390. The HSAS stack was implemented because the TCP/IP component's stack was not sufficient for some large client/server environments. Much work has gone into improving the TCP/IP stack. IBM's goal is to converge to 1 stack when the TCP/IP stack can match the performance of the HSAS stack.