

# Making Assembler Cool Again with z/OS UNIX

Vit Gottwald – CA Technologies  
vit.gottwald@ca.com

SHARE in Anaheim

August 9, 2012

# Outline of Part I

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - Calling a z/OS UNIX service
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - Multiprocess environment
  - Being called by another z/OS UNIX program

# Outline of Part II

- 4 The cool stuff
  - Linking to your old code
  - Reading spool from a z/OS UNIX program
  
- 5 Summary
  - Summary
  - Further reading

# Part I

## The bits and bytes

## Warning: User Experience Session

- I learned this by playing with z/OS UNIX.
- Later I found *some* of the pieces described in manuals and redbooks.
- The current z/OS release at the time of writing this presentation was 1.13.

# Outline

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - Calling a z/OS UNIX service
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - Multiprocess environment
  - Being called by another z/OS UNIX program

# Environments where programs run

Whenever writing a program we have to know:

- What parameters and other information are available to your program when it receives control
- What information is your program expected to return to back to it's caller
- What services/operations are available/allowed

This is generally referred to as the **environment** and the first two items are typically called the **linkage conventions**

# Environments where programs run

Whenever writing a program we have to know:

- What parameters and other information are available to your program when it receives control
- What information is your program expected to return to back to it's caller
- What services/operations are available/allowed

This is generally referred to as the **environment** and the first two items are typically called the **linkage conventions**



# Environments where programs run

Whenever writing a program we have to know:

- What parameters and other information are available to your program when it receives control
- What information is your program expected to return to back to it's caller
- What services/operations are available/allowed

This is generally referred to as the **environment** and the first two items are typically called the **linkage conventions**

# Environments where programs run

Whenever writing a program we have to know:

- What parameters and other information are available to your program when it receives control
- What information is your program expected to return to back to it's caller
- What services/operations are available/allowed

This is generally referred to as the **environment** and the first two items are typically called the **linkage conventions**

# Environments where programs run

Whenever writing a program we have to know:

- What parameters and other information are available to your program when it receives control
- What information is your program expected to return to back to it's caller
- What services/operations are available/allowed

This is generally referred to as the **environment** and the first two items are typically called the **linkage conventions**

# Environments where programs run

Whenever writing a program we have to know:

- What parameters and other information are available to your program when it receives control
- What information is your program expected to return to back to it's caller
- What services/operations are available/allowed

This is generally referred to as the **environment** and the first two items are typically called the **linkage conventions**

## Environment examples

For an assembler program under z/OS some of the typical environments are:

- MVS Batch (Started Task or JOB)
- TSO
- CICS
- Language Environment
- Exit Routines - e.g. Timer Exits, Security Exits
- z/OS UNIX

## Environment examples

For an assembler program under z/OS some of the typical environments are:

- MVS Batch (Started Task or JOB)
- TSO
- CICS
- Language Environment
- Exit Routines - e.g. Timer Exits, Security Exits
- z/OS UNIX

## Environment examples

For an assembler program under z/OS some of the typical environments are:

- MVS Batch (Started Task or JOB)
- TSO
- CICS
- Language Environment
- Exit Routines - e.g. Timer Exits, Security Exits
- z/OS UNIX

## Environment examples

For an assembler program under z/OS some of the typical environments are:

- MVS Batch (Started Task or JOB)
- TSO
- CICS
- Language Environment
- Exit Routines - e.g. Timer Exits, Security Exits
- z/OS UNIX



## Environment examples

For an assembler program under z/OS some of the typical environments are:

- MVS Batch (Started Task or JOB)
- TSO
- CICS
- Language Environment
- Exit Routines - e.g. Timer Exits, Security Exits
- z/OS UNIX

## Environment examples

For an assembler program under z/OS some of the typical environments are:

- MVS Batch (Started Task or JOB)
- TSO
- CICS
- Language Environment
- Exit Routines - e.g. Timer Exits, Security Exits
- z/OS UNIX

## Environment examples

For an assembler program under z/OS some of the typical environments are:

- MVS Batch (Started Task or JOB)
- TSO
- CICS
- Language Environment
- Exit Routines - e.g. Timer Exits, Security Exits
- z/OS UNIX

# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly one created
  - Assigned from a pool
  - "Overusing" a single address space

# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly one created
  - Assigned from a pool
  - "Overusing" a single address space

# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly one created
  - Assigned from a pool
  - "Overusing" a single address space

# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly one created
  - Assigned from a pool
  - "Overusing" a single address space

# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly created
  - Assigned from a pool
  - "Overusing" a single address space



# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly one created
  - Assigned from a pool
  - "Overusing" a single address space

# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly one created
  - Assigned from a pool
  - "Overusing" a single address space

# What are some of the interesting differences?

- Linkage conventions
- Services that are available
  - Input/Output services
- Address space management
  - Newly one created
  - Assigned from a pool
  - "Overusing" a single address space

# Outline

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - Calling a z/OS UNIX service
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - Multiprocess environment
  - Being called by another z/OS UNIX program

# Creating/Terminating New Address Spaces

## TSO

- User logs on to TSO → address space created.
- User logs off TSO → address space terminated.

## MVS Batch Started Tasks

- /START operator command issued → new address space created.
- When program ends, address space terminated.

# Creating/Terminating New Address Spaces

## TSO

- User logs on to TSO → address space created.
- User logs off TSO → address space terminated.

## MVS Batch Started Tasks

- /START operator command issued → new address space created.
- When program ends, address space terminated.

# Creating/Terminating New Address Spaces

## TSO

- User logs on to TSO → address space created.
- User logs off TSO → address space terminated.

## MVS Batch Started Tasks

- /START operator command issued → new address space created.
- When program ends, address space terminated.

# Creating/Terminating New Address Spaces

## TSO

- User logs on to TSO → address space created.
- User logs off TSO → address space terminated.

## MVS Batch Started Tasks

- /START operator command issued → new address space created.
- When program ends, address space terminated.



# Creating/Terminating New Address Spaces

## TSO

- User logs on to TSO → address space created.
- User logs off TSO → address space terminated.

## MVS Batch Started Tasks

- /START operator command issued → new address space created.
- When program ends, address space terminated.

# Creating/Terminating New Address Spaces

## TSO

- User logs on to TSO → address space created.
- User logs off TSO → address space terminated.

## MVS Batch Started Tasks

- /START operator command issued → new address space created.
- When program ends, address space terminated.

# Creating/Terminating New Address Spaces

## TSO

- User logs on to TSO → address space created.
- User logs off TSO → address space terminated.

## MVS Batch Started Tasks

- /START operator command issued → new address space created.
- When program ends, address space terminated.

## Selecting Address Space from a Pool

### MVS Batch JOB:

- User writes a JCL to an internal reader (submit)
- JES selects an Address Space from a pool of available INITIATORs
- The program runs in the INITIATOR address space
- Program ends
- Initiator TCB cleans up and prepares the address space for use by another JOB (or jobstep if it was a multi-step JOB)

## "Overusing" a single address space

CICS , CA-Roscoe

- just one address space where all programs have to run

# UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB



## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## UNIX vs MVS terminology

- In the UNIX world there is not a concept of an *address space*, instead there is the concept of a *process*
- Both address spaces and processes are used to provide an environment in which:
  - Programs are started and execute
  - Isolates individual running programs from each other
- A separately dispatchable work unit within
  - an address space is a *Task Control Block (TCB)*
  - a process it is a *thread*
- z/OS UNIX is implemented as an extension of MVS
  - process has to run within an address space
  - thread has to execute within a TCB

## z/OS UNIX address spaces

OMVS – the z/OS UNIX *kernel*, keeps track of UNIX *processes* and provides callable services (*sycalls*)

BPXOINIT – the *initialization process*, the first process created by the system, PID=1, creates and manages *fork initiators*, starts new processes

BPXAS – the *WLM fork initiator* address space houses a UNIX process created by `fork()` or `spawn()` syscall



# z/OS UNIX address spaces

## OMVS

address space  
(kernel)

**OMVS** – the z/OS UNIX *kernel*, keeps track of UNIX *processes* and provides callable services (*sycalls*)

**BPXOINIT** – the *initialization process*, the first process created by the system, PID=1, creates and manages *fork initiators*, starts new processes

**BPXAS** – the *WLM fork initiator* address space houses a UNIX process created by `fork()` or `spawn()` syscall

# z/OS UNIX address spaces

## OMVS

address space  
(kernel)

## BPXOINIT

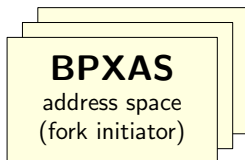
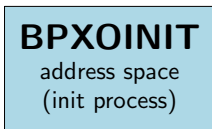
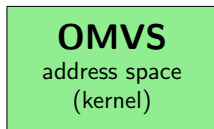
address space  
(init process)

**OMVS** – the z/OS UNIX *kernel*, keeps track of UNIX *processes* and provides callable services (*sycalls*)

**BPXOINIT** – the *initialization process*, the first process created by the system, PID=1, creates and manages *fork initiators*, starts new processes

**BPXAS** – the *WLM fork initiator* address space houses a UNIX process created by `fork()` or `spawn()` syscall

## z/OS UNIX address spaces



**OMVS** – the z/OS UNIX *kernel*, keeps track of UNIX *processes* and provides callable services (*sycalls*)

**BPXOINIT** – the *initialization process*, the first process created by the system, PID=1, creates and manages *fork initiators*, starts new processes

**BPXAS** – the *WLM fork initiator* address space houses a UNIX process created by `fork()` or `spawn()` syscall

## BPXAS address space

- Created by *Workload Manager* if there is a need for a new address space to run a process.
- Reused once the process ends - analogous to how batch initiators are reused.
- Automatically terminated after 30 minutes if there are no `fork()` or `spawn()` request that would require another address space.
- The started procedure is `SYS1.PARMLIB(BPXAS)`
- The message class is `A`. There is no way to dynamically change it.
- When the address space is first started, the job step name is `STEP1`, later when it is reused or `exec()` is called the stepname is changed to `*OMVSEX`

## BPXAS address space

- Created by *Workload Manager* if there is a need for a new address space to run a process.
- Reused once the process ends - analogous to how batch initiators are reused.
- Automatically terminated after 30 minutes if there are no `fork()` or `spawn()` request that would require another address space.
- The started procedure is `SYS1.PARMLIB(BPXAS)`
- The message class is `A`. There is no way to dynamically change it.
- When the address space is first started, the job step name is `STEP1`, later when it is reused or `exec()` is called the stepname is changed to `*OMVSEX`

## BPXAS address space

- Created by *Workload Manager* if there is a need for a new address space to run a process.
- Reused once the process ends - analogous to how batch initiators are reused.
- Automatically terminated after 30 minutes if there are no `fork()` or `spawn()` request that would require another address space.
- The started procedure is `SYS1.PARMLIB(BPXAS)`
- The message class is `A`. There is no way to dynamically change it.
- When the address space is first started, the job step name is `STEP1`, later when it is reused or `exec()` is called the stepname is changed to `*OMVSEX`

## BPXAS address space

- Created by *Workload Manager* if there is a need for a new address space to run a process.
- Reused once the process ends - analogous to how batch initiators are reused.
- Automatically terminated after 30 minutes if there are no `fork()` or `spawn()` request that would require another address space.
- The started procedure is `SYS1.PARMLIB(BPXAS)`
- The message class is `A`. There is no way to dynamically change it.
- When the address space is first started, the job step name is `STEP1`, later when it is reused or `exec()` is called the stepname is changed to `*OMVSEX`

## BPXAS address space

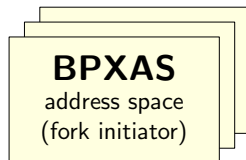
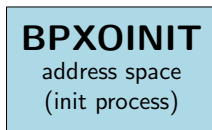
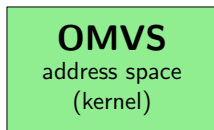
- Created by *Workload Manager* if there is a need for a new address space to run a process.
- Reused once the process ends - analogous to how batch initiators are reused.
- Automatically terminated after 30 minutes if there are no `fork()` or `spawn()` request that would require another address space.
- The started procedure is `SYS1.PARMLIB(BPXAS)`
- The message class is `A`. There is no way to dynamically change it.
- When the address space is first started, the job step name is `STEP1`, later when it is reused or `exec()` is called the stepname is changed to `*OMVSEX`



## BPXAS address space

- Created by *Workload Manager* if there is a need for a new address space to run a process.
- Reused once the process ends - analogous to how batch initiators are reused.
- Automatically terminated after 30 minutes if there are no `fork()` or `spawn()` request that would require another address space.
- The started procedure is `SYS1.PARMLIB(BPXAS)`
- The message class is `A`. There is no way to dynamically change it.
- When the address space is first started, the job step name is `STEP1`, later when it is reused or `exec()` is called the stepname is changed to `*OMVSEX`

# z/OS UNIX address spaces, again



# Outline

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - Calling a z/OS UNIX service
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - Multiprocess environment
  - Being called by another z/OS UNIX program

# Address space $\neq$ process

- Processes form a hierarchical parent-child structure (PID, PPID).
- Address spaces have no such relationship.
- Creating a new process is in theory (and in most implementations of UNIX) a cheap operation.
- Creating an address space is a very expensive operation.
- We may want to access z/OS UNIX services (e.g. TCP/IP) from an already running address space that is not a BPXAS fork initiator.

## Address space $\neq$ process

- Processes form a hierarchical parent-child structure (PID, PPID).
- Address spaces have no such relationship.
- Creating a new process is in theory (and in most implementations of UNIX) a cheap operation.
- Creating an address space is a very expensive operation.
- We may want to access z/OS UNIX services (e.g. TCP/IP) from an already running address space that is not a BPXAS fork initiator.

## Address space $\neq$ process

- Processes form a hierarchical parent-child structure (PID, PPID).
- Address spaces have no such relationship.
- Creating a new process is in theory (and in most implementations of UNIX) a cheap operation.
- Creating an address space is a very expensive operation.
- We may want to access z/OS UNIX services (e.g. TCP/IP) from an already running address space that is not a BPXAS fork initiator.

## Address space $\neq$ process

- Processes form a hierarchical parent-child structure (PID, PPID).
- Address spaces have no such relationship.
- Creating a new process is in theory (and in most implementations of UNIX) a cheap operation.
- Creating an address space is a very expensive operation.
- We may want to access z/OS UNIX services (e.g. TCP/IP) from an already running address space that is not a BPXAS fork initiator.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `_BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (`sh`) (not `tcsh` or `bash`)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are **dubbed** - the kernel registers the address space and it becomes a **process** the TCB becomes a **thread**.



## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `.BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (`sh`) (not `tcsh` or `bash`)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are **dubbed** - the kernel registers the address space and it becomes a **process** the TCB becomes a **thread**.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators

- *Extended attribute 's'* in the filesystem.

If set for a program, new process to run the program can be created within the current address space

- requires `.BPX_SHAREAS=YES`
- Seems to work only from z/OS shell (`sh`) (not `tcsh` or `bash`)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are **dubbed** - the kernel registers the address space and it becomes a **process** the TCB becomes a **thread**.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `_BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (sh) (not tcsh or bash)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are dubbed - the kernel registers the address space and it becomes a process the TCB becomes a thread.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `_BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (sh) (not tcsh or bash)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are dubbed - the kernel registers the address space and it becomes a process the TCB becomes a thread.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `_BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (`sh`) (not `tcsh` or `bash`)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are dubbed - the kernel registers the address space and it becomes a process the TCB becomes a thread.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `_BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (`sh`) (not `tcsh` or `bash`)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are **dubbed** - the kernel registers the address space and it becomes a **process** the TCB becomes a **thread**.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `_BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (`sh`) (not `tcsh` or `bash`)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are **dubbed** - the kernel registers the address space and it becomes a **process** the TCB becomes a **thread**.

## Address Space $\neq$ Process, cont'd

To address these issues:

- Additional control blocks describe a process and a thread, chained from a TCB
- Workload Manager maintains a pool of BPXAS address spaces and reuses them similarly to how JES reuses its initiators
- *Extended attribute 's'* in the filesystem.  
If set for a program, new process to run the program can be created within the current address space
  - requires `_BPX_SHAREAS=YES`
  - Seems to work only from z/OS shell (sh) (not tcsh or bash)
- Any address space can become a process by calling a z/OS UNIX service.

When this happens the address space and the TCB are **dubbed** - the kernel registers the address space and it becomes a **process** the TCB becomes a **thread**.



# Outline

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - **Calling a z/OS UNIX service**
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - Multiprocess environment
  - Being called by another z/OS UNIX program

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].



## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## z/OS UNIX Services (syscalls)

The only way how to get something done in a UNIX system, e.g.

- Open, read from, write to a file
- Create a new process or a thread
- Allocate storage (in z/OS UNIX, use STORAGE macro)

In z/OS UNIX provided as assembler callable services:

- BPX1xxx - AMODE 31 version of service xxx
- BPX2xxx - If exists, a newer version that replaces BXP1xxx
- BPX4xxx - AMODE 64 version of service xxx

Described in great detail in [1].

## Dubbing an address space by calling "Get Process ID"

```
*----- Call Get Process ID service
CALL  BPX1GPI,(PID)
*----- Write To Operator the assigned PID
L     2,PID
CVD  2,DOUBLE
OI   DOUBLE+7,X'0F'
UNPK WTOPID,DOUBLE
LA   1,WTOPRM
WTO  MF=(E,(1))
*----- Abend showing the process and thread IDs
EXRL 0,*
```

(right) click [here](#) to extract the whole program, run it in batch

## Dubbing an address space by calling "Get Process ID"

```
*----- Call Get Process ID service
CALL  BPX1GPI,(PID)

*----- Write To Operator the assigned PID
L     2,PID
CVD  2,DOUBLE
OI   DOUBLE+7,X'0F'
UNPK WTOPID,DOUBLE
LA   1,WTOPRM
WTO  MF=(E,(1))

*----- Abend showing the process and thread IDs
EXRL 0,*
```

(right) click [here](#) to extract the whole program, run it in batch

## Dubbing an address space by calling "Get Process ID"

```
*----- Call Get Process ID service
CALL  BPX1GPI,(PID)
*----- Write To Operator the assigned PID
L     2,PID
CVD  2,DOUBLE
OI   DOUBLE+7,X'OF'
UNPK WTOPID,DOUBLE
LA   1,WTOPRM
WTO  MF=(E,(1))
*----- Abend showing the process and thread IDs
EXRL 0,*
```

(right) click [here](#) to extract the whole program, run it in batch

## Dubbing an address space by calling "Get Process ID"

```
*----- Call Get Process ID service
CALL  BPX1GPI,(PID)
*----- Write To Operator the assigned PID
L     2,PID
CVD  2,DOUBLE
OI   DOUBLE+7,X'OF'
UNPK WTOPID,DOUBLE
LA   1,WTOPRM
WTO  MF=(E,(1))
*----- Abend showing the process and thread IDs
EXRL 0,*
```

(right) click [here](#) to extract the whole program, run it in batch

## Dubbing an address space by calling "Get Process ID"

```
*----- Call Get Process ID service
CALL  BPX1GPI,(PID)
*----- Write To Operator the assigned PID
L     2,PID
CVD  2,DOUBLE
OI   DOUBLE+7,X'OF'
UNPK WTOPID,DOUBLE
LA   1,WTOPRM
WTO  MF=(E,(1))
*----- Abend showing the process and thread IDs
EXRL 0,*
```

(right) click [here](#) to extract the whole program, run it in batch

## Dubbing an address space - JESlog

```
IEF403I Y8VSMPL - STARTED - TIME=15.58.36
+PID 0067109079
+PPID 0000000001
IEA995I SYMPTOM DUMP OUTPUT 443
SYSTEM COMPLETION CODE=0C3 REASON CODE=00000003
TIME=15.58.37 SEQ=38565 CPU=0000 ASID=0165
PSW AT TIME OF ERROR 078D1000 B2800F6C ILC 6 INTC 03
ACTIVE LOAD MODULE ADDRESS=32800F20 OFFSET=0000004C
NAME=GO
DATA AT PSW 32800F66 - C6000000 000058D0 D0045020
AR/GR 0: 00000000/00000000_00011000 1: 00000000/00000000_588206E2
2: FFFFFFFF/FFFFFFF0_040000D7 3: FFFFFFFF/FFFFFFF0_00000001
4: FFFFFFFF/FFFFFFF0_007D89B0 5: FFFFFFFF/FFFFFFF0_007FF358
...
E: 00000000/00000000_B2800F4E F: 00000000/00000000_00000000
END OF SYMPTOM DUMP
BPXP018I THREAD 3304C20000000000, IN PROCESS 67109079, ENDED 444
WITHOUT BEING UNDUBBED WITH COMPLETION CODE 940C3000
```



## Dubbing an address space - JESlog

```
IEF403I Y8VSMPL - STARTED - TIME=15.58.36
+PID 0067109079
+PPID 0000000001
IEA995I SYMPTOM DUMP OUTPUT 443
SYSTEM COMPLETION CODE=0C3 REASON CODE=00000003
TIME=15.58.37 SEQ=38565 CPU=0000 ASID=0165
PSW AT TIME OF ERROR 078D1000 B2800F6C ILC 6 INTC 03
ACTIVE LOAD MODULE ADDRESS=32800F20 OFFSET=0000004C
NAME=GO
DATA AT PSW 32800F66 - C6000000 000058D0 D0045020
AR/GR 0: 00000000/00000000_00011000 1: 00000000/00000000_588206E2
2: FFFFFFFF/FFFFFFF0_040000D7 3: FFFFFFFF/FFFFFFF0_00000001
4: FFFFFFFF/FFFFFFF0_007D89B0 5: FFFFFFFF/FFFFFFF0_007FF358
...
E: 00000000/00000000_B2800F4E F: 00000000/00000000_00000000
END OF SYMPTOM DUMP
BPXP018I THREAD 3304C20000000000, IN PROCESS 67109079, ENDED 444
WITHOUT BEING UNDUBBED WITH COMPLETION CODE 940C3000
```

## Dubbing an address space - JESlog

```
IEF403I Y8VSMPL - STARTED - TIME=15.58.36
+PID 0067109079
+PPID 000000001
IEA995I SYMPTOM DUMP OUTPUT 443
SYSTEM COMPLETION CODE=0C3 REASON CODE=00000003
TIME=15.58.37 SEQ=38565 CPU=0000 ASID=0165
PSW AT TIME OF ERROR 078D1000 B2800F6C ILC 6 INTC 03
ACTIVE LOAD MODULE ADDRESS=32800F20 OFFSET=0000004C
NAME=GO
DATA AT PSW 32800F66 - C6000000 000058D0 D0045020
AR/GR 0: 00000000/00000000_00011000 1: 00000000/00000000_588206E2
2: FFFFFFFF/FFFFFFF_040000D7 3: FFFFFFFF/FFFFFFF_00000001
4: FFFFFFFF/FFFFFFF_007D89B0 5: FFFFFFFF/FFFFFFF_007FF358
...
E: 00000000/00000000_B2800F4E F: 00000000/00000000_00000000
END OF SYMPTOM DUMP
BPXP018I THREAD 3304C20000000000, IN PROCESS 67109079, ENDED 444
WITHOUT BEING UNDUBBED WITH COMPLETION CODE 940C3000
```

## Dubbing an address space - JESlog

```
IEF403I Y8VSMPL - STARTED - TIME=15.58.36
+PID 0067109079
+PPID 000000001
IEA995I SYMPTOM DUMP OUTPUT 443
SYSTEM COMPLETION CODE=0C3 REASON CODE=00000003
TIME=15.58.37 SEQ=38565 CPU=0000 ASID=0165
PSW AT TIME OF ERROR 078D1000 B2800F6C ILC 6 INTC 03
ACTIVE LOAD MODULE ADDRESS=32800F20 OFFSET=0000004C
NAME=GO
DATA AT PSW 32800F66 - C6000000 000058D0 D0045020
AR/GR 0: 00000000/00000000_00011000 1: 00000000/00000000_588206E2
2: FFFFFFFF/FFFFFFF_040000D7 3: FFFFFFFF/FFFFFFF_00000001
4: FFFFFFFF/FFFFFFF_007D89B0 5: FFFFFFFF/FFFFFFF_007FF358
...
E: 00000000/00000000_B2800F4E F: 00000000/00000000_00000000
END OF SYMPTOM DUMP
BPXP018I THREAD 3304C2000000000, IN PROCESS 67109079, ENDED 444
WITHOUT BEING UNDUBBED WITH COMPLETION CODE 940C3000
```

## Calling a z/OS UNIX service

There are several ways how to call a z/OS UNIX service:

- Linking your program with `SYS1.CSSLIB` which provides all the *linkage stubs* (sometimes called the *service stubs*)
- Dynamically loading the stub using `LOAD EP=service name`
- Directly branching to an address provided in [1], Appendix A

LLGT	15,16	CVT (both AMODE 31 and 64)
L	15,544(15)	CSRTABLE
L	15,24(15)	CSR slot
L	15,offset(15)	Address of the service
BALR	14,15	Branch and link

BPX1xxx and BPX2xxx have to be entered in AMODE 31.  
If you want to use 64-bit versions (BPX4xxx), see [1], Using callable services in a 64-bit environment.

## Calling a z/OS UNIX service

There are several ways how to call a z/OS UNIX service:

- Linking your program with SYS1.CSSLIB which provides all the *linkage stubs* (sometimes called the *service stubs*)
- Dynamically loading the stub using LOAD EP=**service name**
- Directly branching to an address provided in [1], Appendix A

LLGT	15,16	CVT (both AMODE 31 and 64)
L	15,544(15)	CSRTABLE
L	15,24(15)	CSR slot
L	15, <b>offset</b> (15)	Address of the service
BALR	14,15	Branch and link

BPX1xxx and BPX2xxx have to be entered in AMODE 31.  
If you want to use 64-bit versions (BPX4xxx), see [1], Using callable services in a 64-bit environment.

## Calling a z/OS UNIX service

There are several ways how to call a z/OS UNIX service:

- Linking your program with SYS1.CSSLIB which provides all the *linkage stubs* (sometimes called the *service stubs*)
- Dynamically loading the stub using LOAD EP=**service name**
- Directly branching to an address provided in [1], Appendix A

LLGT	15,16	CVT (both AMODE 31 and 64)
L	15,544(15)	CSRTABLE
L	15,24(15)	CSR slot
L	15, <b>offset</b> (15)	Address of the service
BALR	14,15	Branch and link

BPX1xxx and BPX2xxx have to be entered in AMODE 31.  
If you want to use 64-bit versions (BPX4xxx), see [1], Using callable services in a 64-bit environment.

## Calling a z/OS UNIX service

There are several ways how to call a z/OS UNIX service:

- Linking your program with SYS1.CSSLIB which provides all the *linkage stubs* (sometimes called the *service stubs*)
- Dynamically loading the stub using LOAD EP=**service name**
- Directly branching to an address provided in [1], Appendix A

LLGT	15,16	CVT (both AMODE 31 and 64)
L	15,544(15)	CSRTABLE
L	15,24(15)	CSR slot
L	15, <b>offset</b> (15)	Address of the service
BALR	14,15	Branch and link

BPX1xxx and BPX2xxx have to be entered in AMODE 31.  
If you want to use 64-bit versions (BPX4xxx), see [1], Using callable services in a 64-bit environment.

## Calling a z/OS UNIX service

There are several ways how to call a z/OS UNIX service:

- Linking your program with SYS1.CSSLIB which provides all the *linkage stubs* (sometimes called the *service stubs*)
- Dynamically loading the stub using LOAD EP=**service name**
- Directly branching to an address provided in [1], Appendix A

LLGT	15,16	CVT (both AMODE 31 and 64)
L	15,544(15)	CSRTABLE
L	15,24(15)	CSR slot
L	15, <b>offset</b> (15)	Address of the service
BALR	14,15	Branch and link

BPX1xxx and BPX2xxx have to be entered in AMODE 31.

If you want to use 64-bit versions (BPX4xxx), see [1], Using callable services in a 64-bit environment.



## Calling a z/OS UNIX service

There are several ways how to call a z/OS UNIX service:

- Linking your program with SYS1.CSSLIB which provides all the *linkage stubs* (sometimes called the *service stubs*)
- Dynamically loading the stub using LOAD EP=**service name**
- Directly branching to an address provided in [1], Appendix A

LLGT	15,16	CVT (both AMODE 31 and 64)
L	15,544(15)	CSRTABLE
L	15,24(15)	CSR slot
L	15, <b>offset</b> (15)	Address of the service
BALR	14,15	Branch and link

BPX1xxx and BPX2xxx have to be entered in AMODE 31.  
If you want to use 64-bit versions (BPX4xxx), see [1], Using callable services in a 64-bit environment.

# Outline

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - Calling a z/OS UNIX service
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - Multiprocess environment
  - Being called by another z/OS UNIX program

## z/OS UNIX syscal parameter list

- We called "Get Process ID" simply by  
CALL BPX1PGI, (PID)
- However, this is rather an exception to the standard call  
format:

```
CALL  ServiceName, (Parm1,      X
      Parm2,              X
      .                   X
      .                   X
      ReturnValue,        X
      ReturnCode,         X
      ReasonCode)
```

## z/OS UNIX syscal parameter list

- We called "Get Process ID" simply by  
CALL BPX1PGI, (PID)
- However, this is rather an exception to the standard call  
format:

```
CALL  ServiceName, (Parm1,      X
                    Parm2,      X
                    .           X
                    .           X
                    ReturnValue,  X
                    ReturnCode,  X
                    ReasonCode)
```

## z/OS UNIX syscal parameter list

- We called "Get Process ID" simply by  
CALL BPX1PGI, (PID)
- However, this is rather an exception to the standard call  
format:

```
CALL  ServiceName, (Parm1,      X
                    Parm2,      X
                    .           X
                    .           X
                    ReturnValue,  X
                    ReturnCode,  X
                    ReasonCode)
```

## z/OS UNIX syscal parameter list

- We called "Get Process ID" simply by  
CALL BPX1PGI, (PID)
- However, this is rather an exception to the standard call  
format:

```
CALL  ServiceName, (Parm1,      X
                    Parm2,      X
                    .            X
                    .            X
                    ReturnValue,  X
                    ReturnCode,  X
                    ReasonCode)
```

## z/OS UNIX syscal parameter list

- We called "Get Process ID" simply by  
CALL BPX1PGI, (PID)
- However, this is rather an exception to the standard call  
format:

```
CALL  ServiceName, (Parm1,      X
                    Parm2,      X
                    .           X
                    .           X
                    ReturnValue,  X
                    ReturnCode, X
                    ReasonCode)
```

## z/OS UNIX Syscal Parameter List

- ServiceName** – name of the service (BPX1EXC, BPX1WRT, ...)
- Parm1,Parm2** – parameters required by the service, mapping macros are described in [1], Appendix B
- ReturnValue** – indicates success or failure of the service (typically 0 means success and -1 failure, but there are exceptions)
- ReturnCode** – POSIX error number (*errno*) returned by the service, see [2], Return codes
- ReasonCode** – reason code further qualifying the return code (has no POSIX equivalent), see [2], Reason codes



## Simple call to write() $\equiv$ BPX1WRT

```
CALL  BPX1WRT, (FileDescriptor,      X
              BufferAddress,         X
              BufferALET,            X
              WriteCount,           X
              ReturnValue,           X
              ReturnCode,           X
              ReasonCode)
```

FileDescriptor – (fullword), 0=stdin, 1=stdout, 2=stderr

BufferAddress – Starting address of data to be written (fullword)

BufferALET – ALET of the buffer (fullword), normally zero (0)

WriteCount – Number of bytes to be written

## Simple call to write() $\equiv$ BPX1WRT

```
CALL BPX1WRT, (FileDescriptor,      X
               BufferAddress,        X
               BufferALET,           X
               WriteCount,          X
               ReturnValue,          X
               ReturnCode,          X
               ReasonCode)
```

FileDescriptor – (fullword), 0=stdin, 1=stdout, 2=stderr

BufferAddress – Starting address of data to be written (fullword)

BufferALET – ALET of the buffer (fullword), normally zero (0)

WriteCount – Number of bytes to be written

## Simple call to write() $\equiv$ BPX1WRT

```
CALL BPX1WRT, (FileDescriptor,      X
               BufferAddress,        X
               BufferALET,           X
               WriteCount,           X
               ReturnValue,          X
               ReturnCode,          X
               ReasonCode)
```

**FileDescriptor** – (fullword), 0=stdin, 1=stdout, 2=stderr

**BufferAddress** – Starting address of data to be written (fullword)

**BufferALET** – ALET of the buffer (fullword), normally zero (0)

**WriteCount** – Number of bytes to be written

# Parameter list for BPX1WRT

# Parameter list for BPX1WRT

R1

@Parm List

# Parameter list for BPX1WRT

R1

@Parm List

Parm List  
(31-bit pointers)

@PARAM1

@PARAM2

@PARAM3

@PARAM4

@Return Value

@Return Code

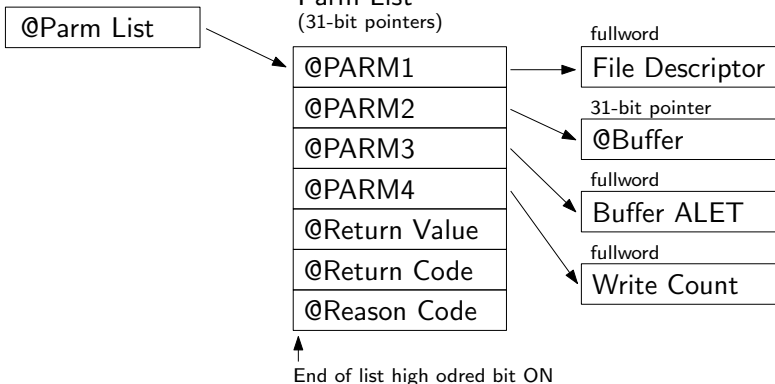
@Reason Code



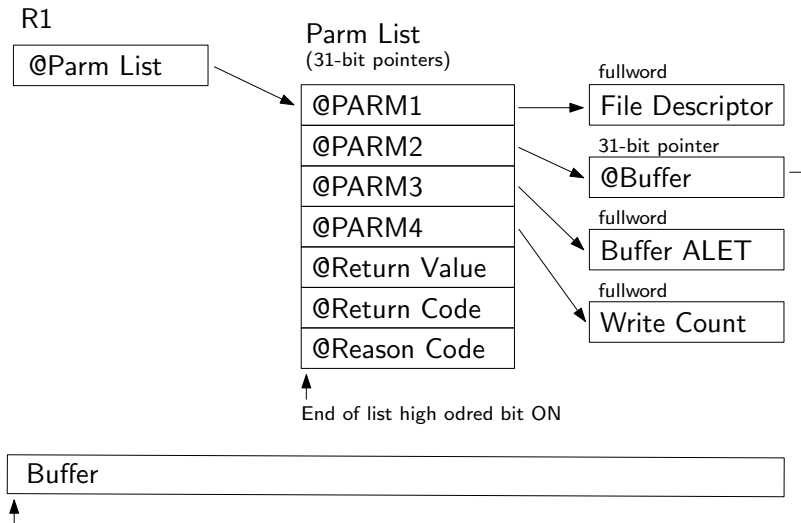
End of list high order bit ON

# Parameter list for BPX1WRT

R1

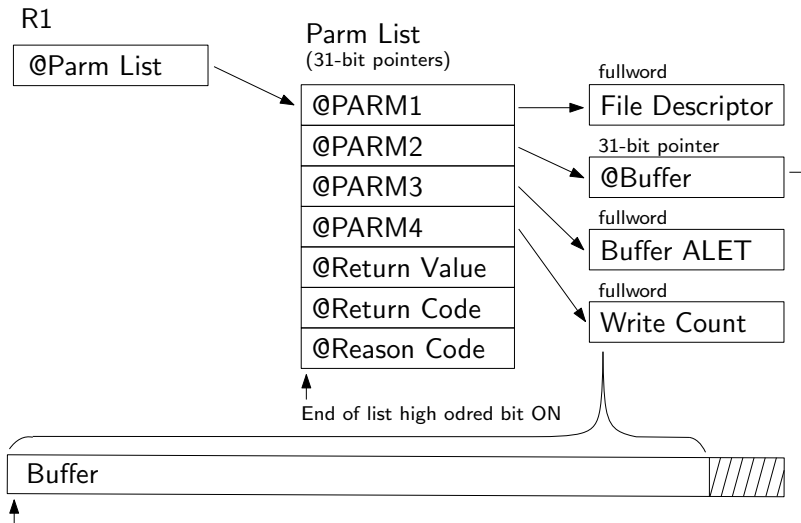


# Parameter list for BPX1WRT

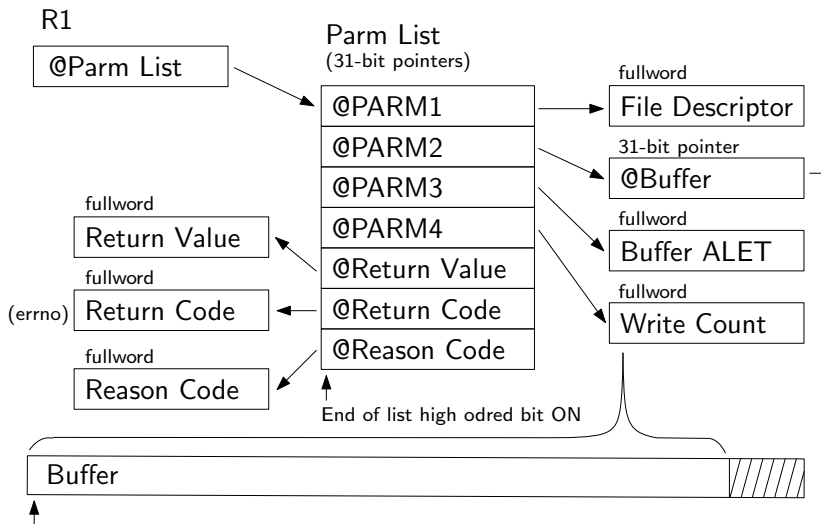




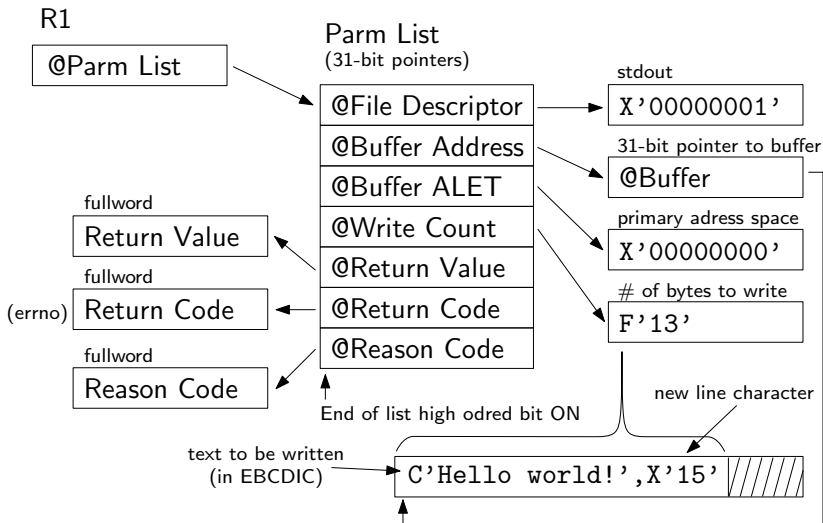
## Parameter list for BPX1WRT



# Parameter list for BPX1WRT



# Parameter list to print "Hello world!" to STDOUT



# z/OS UNIX Assembler "Hello world!"

```
...
*
*----- Write message to standard output
*
          CALL  BPX1WRT,(FD,AMSG,ALET,LEN,RTN_VAL,RTN_COD,RSN_COD),VL
...
FD          DC      F'1'           File Descriptor (1=STDOUT,2=STDERR)
AMSG        DC      A(MSG)         @Message
MSG         DC      C'Hello world!' Message to print
NEW_LINE    DC      X'15'         EBCDIC New Line Character
ALET        DC      F'0'           Alet For The Message (0 - DEFAULT)
LEN         DC      A(L'MSG+L'NEW_LINE) How Many Bytes To Write
RTN_VAL     DC      F'0'           Return Value
RTN_COD     DC      F'0'           Return Code
RSN_COD     DC      F'0'           Reason Code
...

```

(right) click [here](#) to extract the whole program, run it from shell

# z/OS UNIX Assembler "Hello world!"

```
...
*
*----- Write message to standard output
*
          CALL  BPX1WRT,(FD,AMSG,ALET,LEN,RTN_VAL,RTN_COD,RSN_COD),VL
...
FD      DC      F'1'           File Descriptor (1=STDOUT,2=STDERR)
AMSG     DC      A(MSG)        @Message
MSG      DC      C'Hello world!' Message to print
NEW_LINE DC      X'15'        EBCDIC New Line Character
ALET     DC      F'0'         Alet For The Message (0 - DEFAULT)
LEN      DC      A(L'MSG+L'NEW_LINE) How Many Bytes To Write
RTN_VAL  DC      F'0'         Return Value
RTN_COD  DC      F'0'         Return Code
RSN_COD  DC      F'0'         Reason Code
...

```

(right) click [here](#) to extract the whole program, run it from shell

# z/OS UNIX Assembler "Hello world!"

```
...
*
*----- Write message to standard output
*
          CALL  BPX1WRT,(FD,AMSG,ALET,LEN,RTN_VAL,RTN_COD,RSN_COD),VL
...
FD          DC      F'1'           File Descriptor (1=STDOUT,2=STDERR)
AMSG        DC      A(MSG)         @Message
MSG         DC      C'Hello world!' Message to print
NEW_LINE    DC      X'15'         EBCDIC New Line Character
ALET        DC      F'0'           Alet For The Message (0 - DEFAULT)
LEN         DC      A(L'MSG+L'NEW_LINE) How Many Bytes To Write
RTN_VAL     DC      F'0'           Return Value
RTN_COD     DC      F'0'           Return Code
RSN_COD     DC      F'0'           Reason Code
...

```

(right) click [here](#) to extract the whole program, run it from shell

# z/OS UNIX Assembler "Hello world!"

```
...
*
*----- Write message to standard output
*
          CALL  BPX1WRT,(FD,AMSG,ALET,LEN,RTN_VAL,RTN_COD,RSN_COD),VL
...
FD          DC      F'1'           File Descriptor (1=STDOUT,2=STDERR)
AMSG        DC      A(MSG)         @Message
MSG         DC      C'Hello world!' Message to print
NEW_LINE    DC      X'15'         EBCDIC New Line Character
ALET       DC      F'0'           Alet For The Message (0 - DEFAULT)
LEN         DC      A(L'MSG+L'NEW_LINE) How Many Bytes To Write
RTN_VAL     DC      F'0'           Return Value
RTN_COD     DC      F'0'           Return Code
RSN_COD     DC      F'0'           Reason Code
...

```

(right) click [here](#) to extract the whole program, run it from shell

# z/OS UNIX Assembler "Hello world!"

```
...
*
*----- Write message to standard output
*
          CALL  BPX1WRT,(FD,AMSG,ALET,LEN,RTN_VAL,RTN_COD,RSN_COD),VL
...
FD          DC      F'1'           File Descriptor (1=STDOUT,2=STDERR)
AMSG        DC      A(MSG)         @Message
MSG         DC      C'Hello world!' Message to print
NEW_LINE    DC      X'15'         EBCDIC New Line Character
ALET        DC      F'0'           Alet For The Message (0 - DEFAULT)
LEN        DC      A(L'MSG+L'NEW_LINE) How Many Bytes To Write
RTN_VAL     DC      F'0'           Return Value
RTN_COD     DC      F'0'           Return Code
RSN_COD     DC      F'0'           Reason Code
...

```

(right) click [here](#) to extract the whole program, run it from shell



# z/OS UNIX Assembler "Hello world!"

```
...
*
*----- Write message to standard output
*
          CALL  BPX1WRT,(FD,AMSG,ALET,LEN,RTN_VAL,RTN_COD,RSN_COD),VL
...
FD          DC      F'1'           File Descriptor (1=STDOUT,2=STDERR)
AMSG        DC      A(MSG)         @Message
MSG         DC      C'Hello world!' Message to print
NEW_LINE    DC      X'15'         EBCDIC New Line Character
ALET        DC      F'0'           Alet For The Message (0 - DEFAULT)
LEN         DC      A(L'MSG+L'NEW_LINE) How Many Bytes To Write
RTN_VAL     DC      F'0'           Return Value
RTN_COD     DC      F'0'           Return Code
RSN_COD     DC      F'0'           Reason Code
...

```

(right) click [here](#) to extract the whole program, run it from shell

# z/OS UNIX Assembler "Hello world!"

```
...
*
*----- Write message to standard output
*
          CALL  BPX1WRT,(FD,AMSG,ALET,LEN,RTN_VAL,RTN_COD,RSN_COD),VL
...
FD          DC      F'1'           File Descriptor (1=STDOUT,2=STDERR)
AMSG        DC      A(MSG)         @Message
MSG         DC      C'Hello world!' Message to print
NEW_LINE    DC      X'15'         EBCDIC New Line Character
ALET        DC      F'0'           Alet For The Message (0 - DEFAULT)
LEN         DC      A(L'MSG+L'NEW_LINE) How Many Bytes To Write
RTN_VAL     DC      F'0'           Return Value
RTN_COD     DC      F'0'           Return Code
RSN_COD     DC      F'0'           Reason Code
...

```

(right) click [here](#) to extract the whole program, run it from shell

# Outline

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - Calling a z/OS UNIX service
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - **Multiprocess environment**
  - Being called by another z/OS UNIX program

## Even Simpler Hello World!

```
WTOSMPL  CSECT
WTOSMPL  AMODE 31
WTOSMPL  RMODE ANY
          LR    4,14
          WTO   'Hello world!'
          LR    14,4
          LHI   15,0
          BR    14
          END   WTOSMPL
```

## Even Simpler Hello World!

```
WTOSMPL  CSECT  
WTOSMPL  AMODE 31  
WTOSMPL  RMODE ANY  
          LR    4,14  
          WTO   'Hello world!'  
          LR    14,4  
          LHI   15,0  
          BR    14  
          END   WTOSMPL
```

```
$
```

## Even Simpler Hello World!

```
WTOSMPL  CSECT
WTOSMPL  AMODE 31
WTOSMPL  RMODE ANY
          LR    4,14
          WTO   'Hello world!'
          LR    14,4
          LHI   15,0
          BR    14
          END   WTOSMPL
```

```
$ as -o wto.o wto.as
```

## Even Simpler Hello World!

```
WTOSMPL  CSECT
WTOSMPL  AMODE 31
WTOSMPL  RMODE ANY
          LR    4,14
          WTO   'Hello world!'
          LR    14,4
          LHI   15,0
          BR    14
          END   WTOSMPL
```

```
$ as -o wto.o wto.as
Assembler Done No Statements Flagged
$
```

## Even Simpler Hello World!

```
WTOSMPL  CSECT
WTOSMPL  AMODE 31
WTOSMPL  RMODE ANY
          LR    4,14
          WTO   'Hello world!'
          LR    14,4
          LHI   15,0
          BR    14
          END   WTOSMPL
```

```
$ as -o wto.o wto.as
Assembler Done No Statements Flagged
$ ld -o wto wto.o
```



## Even Simpler Hello World!

```
WTOSMPL  CSECT
WTOSMPL  AMODE 31
WTOSMPL  RMODE ANY
          LR    4,14
          WTO   'Hello world!'
          LR    14,4
          LHI   15,0
          BR    14
          END   WTOSMPL
```

```
$ as -o wto.o wto.as
Assembler Done No Statements Flagged
$ ld -o wto wto.o
$
```

# WTO Hello World

```
$ ./wto
```

# WTO Hello World

```
$ ./wto  
$
```

# WTO Hello World

```
$ ./wto  
$ _BPXX_JOBLOG=STDERR ./wto
```

# WTO Hello World

```
$ ./wto  
$ _BPXX_JOBLOG=STDERR ./wto  
13.49.22 STC21558 +Hello world!  
$
```

# WTO Hello World

```
$ ./wto  
$ _BPXX_JOBLOG=STDERR ./wto  
13.49.22 STC21558 +Hello world!  
$ _BPXX_JOBLOG=STDERR ./wto  
13.49.36 STC25682 +Hello world!  
$
```

# WTO Hello World

```
$ ./wto  
$ _BPXX_JOBLOG=STDERR ./wto  
13.49.22 STC21558 +Hello world!  
$ _BPXX_JOBLOG=STDERR ./wto  
13.49.36 STC25682 +Hello world!  
$
```

# WTO Hello World

```
$ ./wto
$ _BPXX_JOBLOG=STDERR ./wto
13.49.22 STC21558 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.36 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.38 STC25682 +Hello world!
$
```



# WTO Hello World

```
$ ./wto
$ _BPXX_JOBLOG=STDERR ./wto
13.49.22 STC21558 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.36 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.38 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR _BPX_SHAREAS=YES ./wto
```

# WTO Hello World

```
$ ./wto
$ _BPXX_JOBLOG=STDERR ./wto
13.49.22 STC21558 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.36 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.38 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR _BPX_SHAREAS=YES ./wto
13.51.21 STC42162 +Hello world!
$
```

# WTO Hello World

```
$ ./wto
$ _BPXX_JOBLOG=STDERR ./wto
13.49.22 STC21558 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.36 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.38 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR _BPX_SHAREAS=YES ./wto
13.51.21 STC42162 +Hello world!
$ _BPXX_JOBLOG=STDERR _BPX_SHAREAS=YES ./wto
13.51.26 STC42162 +Hello world!
$
```

# WTO Hello World

```
$ ./wto
$ _BPXX_JOBLOG=STDERR ./wto
13.49.22 STC21558 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.36 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR ./wto
13.49.38 STC25682 +Hello world!
$ _BPXX_JOBLOG=STDERR _BPX_SHAREAS=YES ./wto
13.51.21 STC42162 +Hello world!
$ _BPXX_JOBLOG=STDERR _BPX_SHAREAS=YES ./wto
13.51.26 STC42162 +Hello world!
$ _BPXX_JOBLOG=STDERR _BPX_SHAREAS=YES ./wto
13.51.28 STC42162 +Hello world!
$
```

# z/OS UNIX Processes/Adress Spaces

```
SYSDUMP VTAM CA11 ----- ACTIVITY, S
Command =====>
-----
(r)   IFA%   IIP%   CP%   ...50..100   -Condi
CPU   0%    12%   83%   ██████████   ENQ
LCPU  0%    10%   82%   ██████████   NORES
                                           NODMP
Spool 48% ██████████
-----
Formats DEFAULT ALERTS CPU GOTVI01 IO
Status  NOSRT  NOLIM  NOSEL  NODST  NOPFX
-----
*
ALL
Cmd  Jobname  Stepname  Type  Jobnr  ASID
---  -
GOTVI011 *OMVSEX  OTX    42162  0275
GOTVI01  STEP1     OTX    42189  02E5
***** End
```

# z/OS UNIX Processes/Address Spaces

```
SYSDUMP VTAM CA11 ----- ACTIVITY, S
Command =====>
-----
(r)   IFA%   IIP%   CP%   ...50..100   -Condi
CPU   0%    12%   83%   ██████████   ENQ
LCPU  0%    10%   82%   ██████████   NORES
                                           NODMP
Spool                                48% ██████████
-----
Formats  DEFAULT ALERTS CPU GOTVI01 IO
Status   NOSRT  NOLIM NOSEL NODST NOPFX
-----
*                                     ALL
Cmd Jobname  stepname  Type   Jobnr  ASID
--- GOTVI011 *OMVSEX   OTX    42162  0275
--- GOTVI01  STEP1      OTX    42189  02E5
***** End
```

```
$ ps -A -o jobname,xasid,args
JOBNAME  ASID  COMMAND
GOTVI011 275  sh
GOTVI01  2e5  otelnetd -Y GOTVI01W7.ca.com -p ...
GOTVI012 1ec  ps -o jobname,xasid,args
$
```

# Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

# Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out



# Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out



## Multiprocess environment

Environment with multiple z/OS UNIX processes in one address space:

- Enabled by `_BPX_SHARE_AS=YES` environment variable
- Each process has a different MVS identity (its own process-level ACCE attached at TCBSENV)
- Programs that change security environment not allowed
- Certain services restricted to prevent user with one MVS identity from
  - affecting all the other processes in the address space
  - creating a new process with a different identity
- *Shared address space* file attribute required (by default on)
  - `ls -E program_name` to find out

```
$ ls -E wto
-rwxrwxrwx  --s-  1 GOTVIO1  OMVSRP      4096 Jul 31 05:34 wto
```

# Outline

- 1 Environments where programs run
  - General overview
  - Address space management
  - Address space vs process
- 2 z/OS UNIX assembler services
  - Calling a z/OS UNIX service
  - Parameter list for z/OS UNIX service
- 3 Executing a program from z/OS UNIX shell
  - Multiprocess environment
  - Being called by another z/OS UNIX program

# Registers contents

- Not explicitly described in [1],
- Seems to follow the standard linkage conventions as described in [4] (both for AMODE 31 and 64)
- On entry:
  - R1 points to a parameter list
  - R13 points to a save area
  - R14 points to return address (where is an SVC 3 instruction)
  - R15 points to the entry point
- On return:
  - R15 contains the return code (echo \$?)
  - R14 contains the adress where to return via BR 14

## Registers contents

- Not explicitly described in [1],
- Seems to follow the standard linkage conventions as described in [4] (both for AMODE 31 and 64)
- On entry:
  - R1 points to a parameter list
  - R13 points to a save area
  - R14 points to return address (where is an SVC 3 instruction)
  - R15 points to the entry point
- On return:
  - R15 contains the return code (echo \$?)
  - R14 contains the adress where to return via BR 14

## Registers contents

- Not explicitly described in [1],
- Seems to follow the standard linkage conventions as described in [4] (both for AMODE 31 and 64)
- On entry:
  - R1 points to a parameter list
  - R13 points to a save area
  - R14 points to return address (where is an SVC 3 instruction)
  - R15 points to the entry point
- On return:
  - R15 contains the return code (echo \$?)
  - R14 contains the adress where to return via BR 14

## Registers contents

- Not explicitly described in [1],
- Seems to follow the standard linkage conventions as described in [4] (both for AMODE 31 and 64)
- On entry:
  - R1 points to a parameter list
  - R13 points to a save area
  - R14 points to return address (where is an SVC 3 instruction)
  - R15 points to the entry point
- On return:
  - R15 contains the return code (echo \$?)
  - R14 contains the adress where to return via BR 14

## Registers contents

- Not explicitly described in [1],
- Seems to follow the standard linkage conventions as described in [4] (both for AMODE 31 and 64)
- On entry:
  - R1 points to a parameter list
  - R13 points to a save area
  - R14 points to return address (where is an SVC 3 instruction)
  - R15 points to the entry point
- On return:
  - R15 contains the return code (echo \$?)
  - R14 contains the adress where to return via BR 14

## What does the parameter list look like?

Creating a new process using `spawn()` service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,              X
               Argument_count,        X
               Argument_length_list,  X
               Argument_list,         X
               Environment_count,      X
               Environment_data_length, X
               Environment_data_list,  X
               Filedesc_count,        X
               Filedesc_list,         X
               Inherit_area_len,      X
               Inherit_area,         X
               Return_value,          X
               Return_code,           X
               Reason_code)
```



## What does the parameter list look like?

Creating a new process using `spawn()` service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,              X
               Argument_count,        X
               Argument_length_list,   X
               Argument_list,         X
               Environment_count,      X
               Environment_data_length, X
               Environment_data_list,  X
               Filedesc_count,         X
               Filedesc_list,          X
               Inherit_area_len,      X
               Inherit_area,          X
               Return_value,          X
               Return_code,           X
               Reason_code)
```

## What does the parameter list look like?

Creating a new process using `spawn()` service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,              X
               Argument_count,        X
               Argument_length_list,  X
               Argument_list,         X
               Environment_count,     X
               Environment_data_length, X
               Environment_data_list, X
               Filedesc_count,        X
               Filedesc_list,         X
               Inherit_area_len,     X
               Inherit_area,         X
               Return_value,          X
               Return_code,           X
               Reason_code)
```

## What does the parameter list look like?

Creating a new process using spawn() service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,              X
               Argument_count,        X
               Argument_length_list,  X
               Argument_list,         X
               Environment_count,     X
               Environment_data_length, X
               Environment_data_list, X
               Filedesc_count,        X
               Filedesc_list,         X
               Inherit_area_len,      X
               Inherit_area,          X
               Return_value,          X
               Return_code,           X
               Reason_code)
```

## What does the parameter list look like?

Creating a new process using spawn() service:

```
CALL BPX1SPN, (Pathname_length,      X
              Pathname,              X
              Argument_count,        X
              Argument_length_list,  X
              Argument_list,         X
              Environment_count,     X
              Environment_data_length, X
              Environment_data_list, X
              Filedesc_count,        X
              Filedesc_list,         X
              Inherit_area_len,      X
              Inherit_area,          X
              Return_value,          X
              Return_code,           X
              Reason_code)
```

## What does the parameter list look like?

Creating a new process using spawn() service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,              X
               Argument_count,        X
               Argument_length_list,  X
               Argument_list,         X
               Environment_count,     X
               Environment_data_length, X
               Environment_data_list, X
               Filedesc_count,        X
               Filedesc_list,         X
               Inherit_area_len,      X
               Inherit_area,          X
               Return_value,          X
               Return_code,           X
               Reason_code)
```

## What does the parameter list look like?

Creating a new process using spawn() service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,              X
               Argument_count,        X
               Argument_length_list,  X
               Argument_list,         X
               Environment_count,     X
               Environment_data_length, X
               Environment_data_list, X
               Filedesc_count,        X
               Filedesc_list,         X
               Inherit_area_len,      X
               Inherit_area,          X
               Return_value,          X
               Return_code,           X
               Reason_code)
```

## What does the parameter list look like?

Creating a new process using spawn() service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,             X
               Argument_count,       X
               Argument_length_list,  X
               Argument_list,        X
               Environment_count,     X
               Environment_data_length, X
               Environment_data_list, X
               Filedesc_count,       X
               Filedesc_list,       X
               Inherit_area_len,    X
               Inherit_area,       X
               Return_value,        X
               Return_code,         X
               Reason_code)
```

## What does the parameter list look like?

Creating a new process using spawn() service:

```
CALL BPX1SPN, (Pathname_length,      X
               Pathname,              X
               Argument_count,        X
               Argument_length_list,  X
               Argument_list,         X
               Environment_count,     X
               Environment_data_length, X
               Environment_data_list, X
               Filedesc_count,        X
               Filedesc_list,         X
               Inherit_area_len,      X
               Inherit_area,          X
               Return_value,          X
               Return_code,           X
               Reason_code)
```



Environments where programs run  
z/OS UNIX assembler services  
Executing a program from z/OS UNIX shell

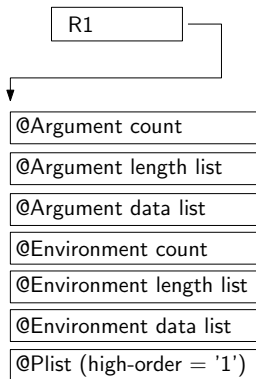
Multiprocess environment  
Being called by another z/OS UNIX program

# What does the parameter list look like?, cont'd

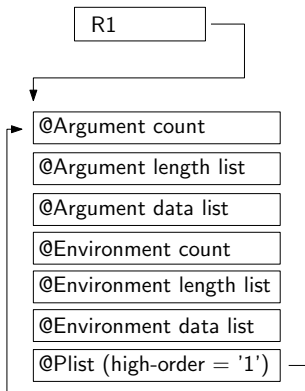
# What does the parameter list look like?, cont'd

R1

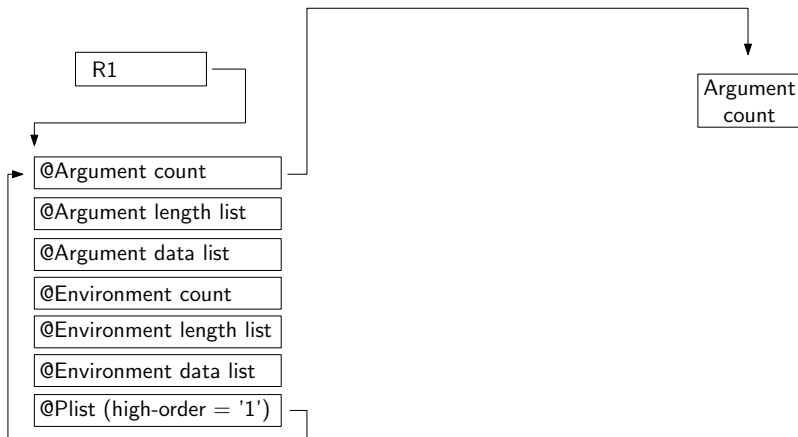
# What does the parameter list look like?, cont'd



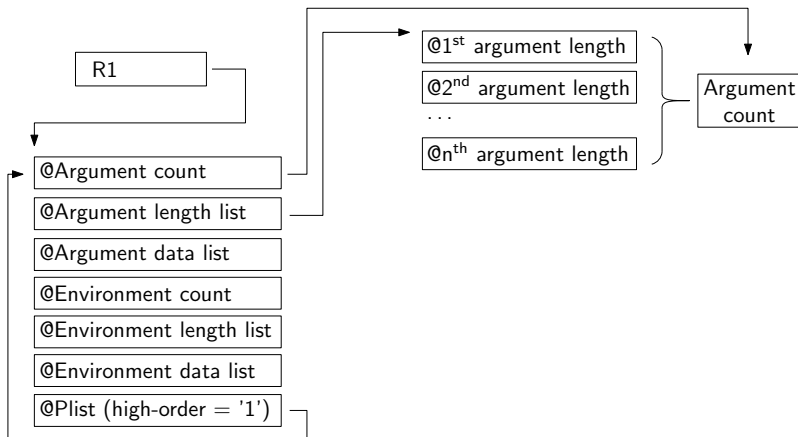
# What does the parameter list look like?, cont'd



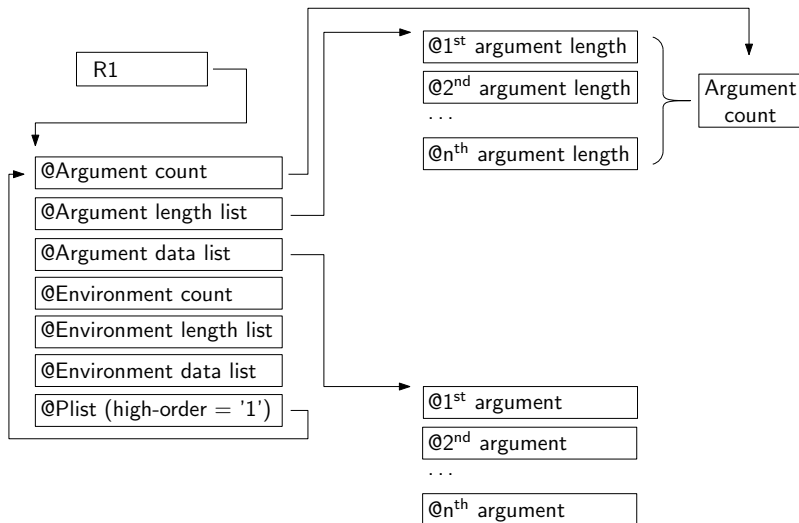
# What does the parameter list look like?, cont'd



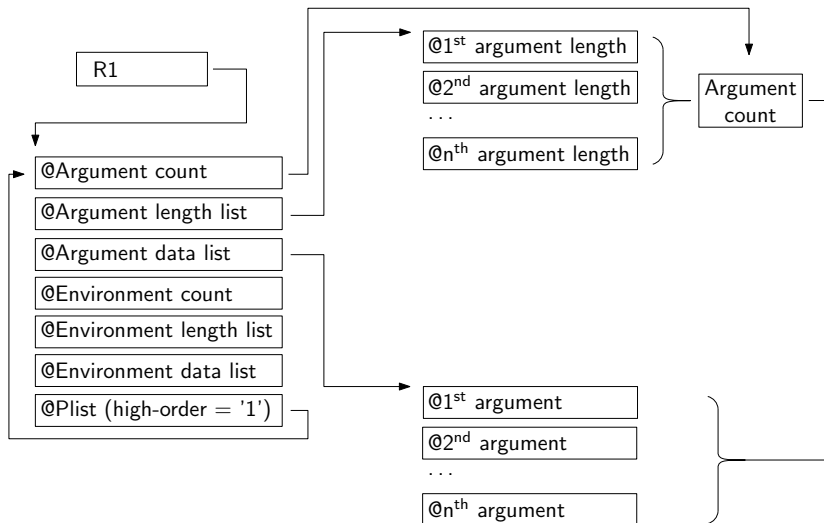
## What does the parameter list look like?, cont'd



# What does the parameter list look like?, cont'd

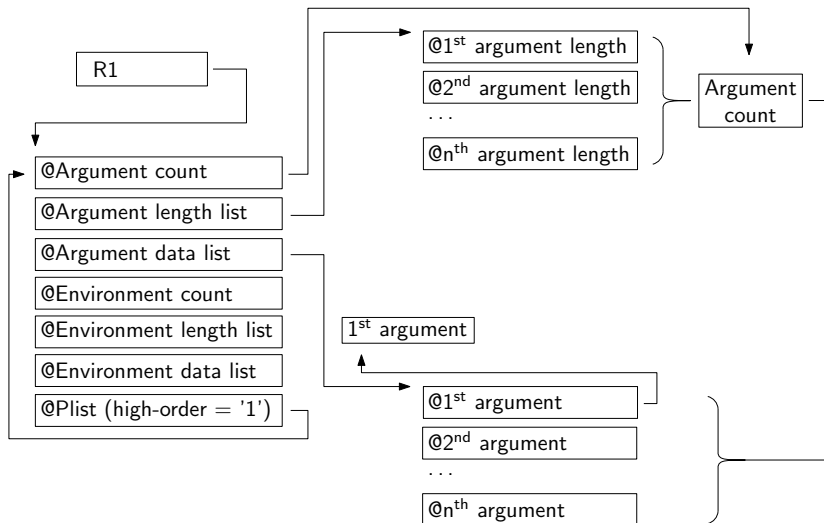


## What does the parameter list look like?, cont'd

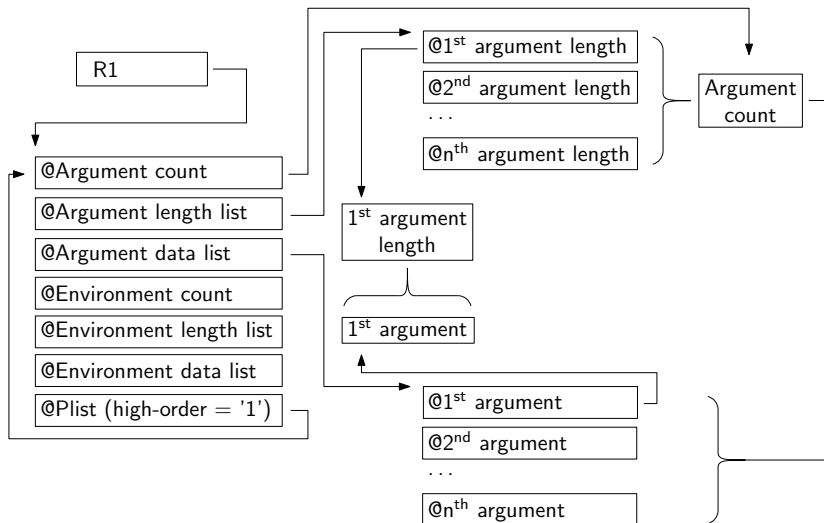




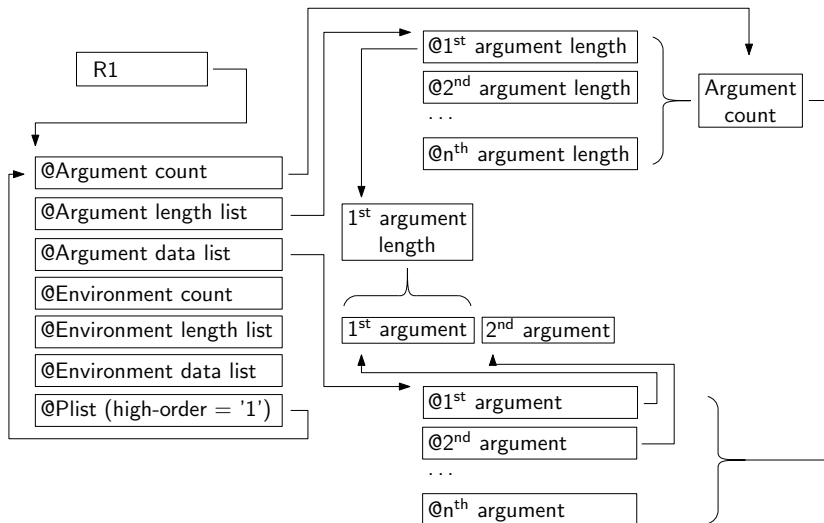
# What does the parameter list look like?, cont'd



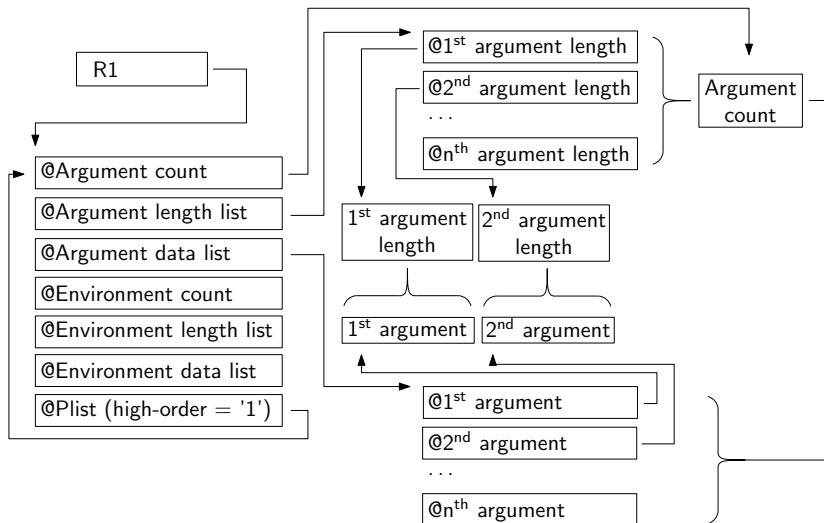
## What does the parameter list look like?, cont'd



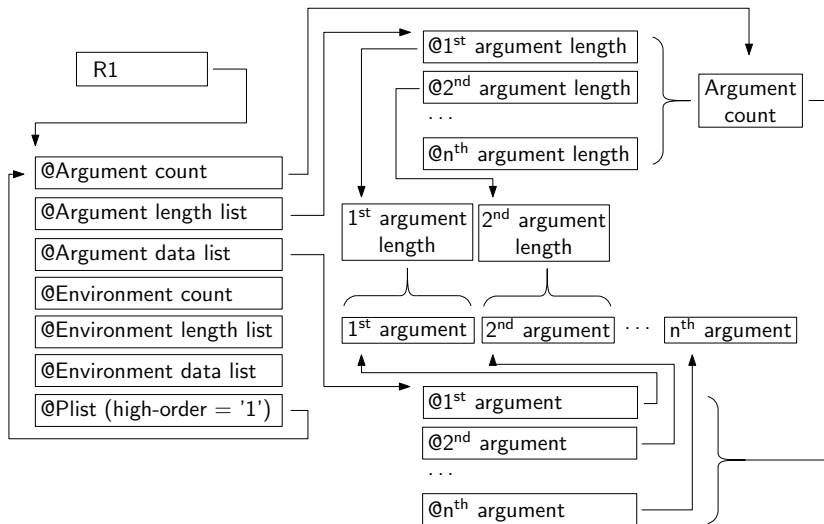
# What does the parameter list look like?, cont'd



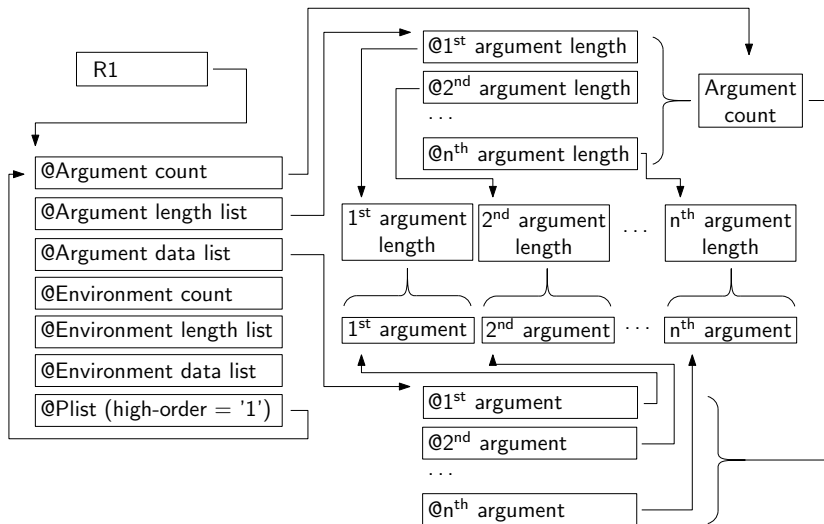
# What does the parameter list look like?, cont'd



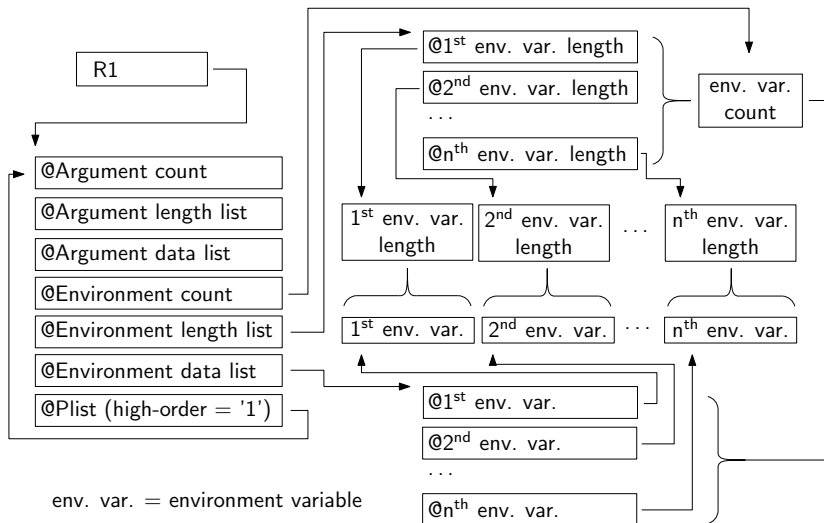
# What does the parameter list look like?, cont'd



# What does the parameter list look like?, cont'd



# What does the parameter list look like?, cont'd



# Printing Parameters To Standard Output

```
LR    6,1           @Param List
L     7,0(,6)       @Argument Count
L     7,0(,7)       Argument Count
*----- Print parameters
L     2,4(,6)       @Argument Length List
L     3,8(,6)       @Argument Data List
LOOP  DC    0H
L     4,0(,2)       @Argument Length
L     5,0(,3)       @Argument Data
L     4,0(,4)       Argumen Length
LA    1,MSG(4)
MVI   0(1),NEW_LINE New Line Char After The Argument
BCTR  4,0
EXRL  4,#+6        Copy Argument To The Message
MVC   MSG(0),0(5)
AHI   4,2
ST    4,MSG_LEN    Save Message Length
CALL  BPX1WRT,MF=(E,SYSCWRT) Print The Message (Argument)
*
LA    2,4(,2)       @Next Argument Length
LA    3,4(,3)       @Next Arugment Data
JCT   7,LOOP
```

(right) click [here](#) to extract the whole program



# Printing Parameters To Standard Output

```
LR      6,1           @Param List
L       7,0(,6)      @Argument Count
L       7,0(,7)      Argument Count
*----- Print parameters
L       2,4(,6)      @Argument Length List
L       3,8(,6)      @Argument Data List
LOOP    DC          0H
L       4,0(,2)      @Argument Length
L       5,0(,3)      @Argument Data
L       4,0(,4)      Argumen Length
LA      1,MSG(4)
MVI     0(1),NEW_LINE  New Line Char After The Argument
BCTR   4,0
EXRL   4,#+6        Copy Argument To The Message
MVC    MSG(0),0(5)
AHI    4,2
ST     4,MSG_LEN    Save Message Length
CALL   BPX1WRT,MF=(E,SYSCWRT)  Print The Message (Argument)
*
LA     2,4(,2)      @Next Argument Length
LA     3,4(,3)      @Next Arugment Data
JCT    7,LOOP
```

(right) click [here](#) to extract the whole program

# Printing Parameters To Standard Output

```
LR    6,1           @Param List
L     7,0(,6)       @Argument Count
L     7,0(,7)       @Argument Count
*----- Print parameters
L     2,4(,6)       @Argument Length List
L     3,8(,6)       @Argument Data List
LOOP  DC    0H
L     4,0(,2)       @Argument Length
L     5,0(,3)       @Argument Data
L     4,0(,4)       @Argument Length
LA    1,MSG(4)
MVI   0(1),NEW_LINE New Line Char After The Argument
BCTR  4,0
EXRL  4,#+6        Copy Argument To The Message
MVC   MSG(0),0(5)
AHI   4,2
ST    4,MSG_LEN    Save Message Length
CALL  BPX1WRT,MF=(E,SYSCWRT) Print The Message (Argument)
*
LA    2,4(,2)       @Next Argument Length
LA    3,4(,3)       @Next Argument Data
JCT   7,LOOP
```

(right) click [here](#) to extract the whole program

# Printing Parameters To Standard Output

```
LR      6,1           @Param List
L       7,0(,6)      @Argument Count
L       7,0(,7)      @Argument Count
*----- Print parameters
L       2,4(,6)      @Argument Length List
L       3,8(,6)      @Argument Data List
LOOP    DC          0H
L       4,0(,2)      @Argument Length
L       5,0(,3)      @Argument Data
L       4,0(,4)      @Argument Length
LA      1,MSG(4)
MVI     0(1),NEW_LINE New Line Char After The Argument
BCTR   4,0
EXRL   4,#+6        Copy Argument To The Message
MVC    MSG(0),0(5)
AHI    4,2
ST     4,MSG_LEN    Save Message Length
CALL   BPX1WRT,MF=(E,SYSCWRT) Print The Message (Argument)
*
LA     2,4(,2)      @Next Argument Length
LA     3,4(,3)      @Next Argument Data
JCT    7,LOOP
```

(right) click [here](#) to extract the whole program

# Printing Parameters To Standard Output

```
LR      6,1           @Param List
L       7,0(,6)      @Argument Count
L       7,0(,7)      Argument Count
*----- Print parameters
L       2,4(,6)      @Argument Length List
L       3,8(,6)      @Argument Data List
LOOP    DC          0H
L       4,0(,2)      @Argument Length
L       5,0(,3)      @Argument Data
L       4,0(,4)      Argumen Length
LA      1,MSG(4)
MVI     0(1),NEW_LINE  New Line Char After The Argument
BCTR    4,0
EXRL    4,#+6        Copy Argument To The Message
MVC     MSG(0),0(5)
AHI     4,2
ST      4,MSG_LEN    Save Message Length
CALL    BPX1WRT,MF=(E,SYSCWRT) Print The Message (Argument)
*
LA      2,4(,2)      @Next Argument Length
LA      3,4(,3)      @Next Arugment Data
JCT     7,LOOP
```

(right) click [here](#) to extract the whole program

# Printing Parameters To Standard Output

```
LR      6,1           @Param List
L       7,0(,6)      @Argument Count
L       7,0(,7)      Argument Count
*----- Print parameters
L       2,4(,6)      @Argument Length List
L       3,8(,6)      @Argument Data List
LOOP    DC          0H
L       4,0(,2)      @Argument Length
L       5,0(,3)      @Argument Data
L       4,0(,4)      Argumen Length
LA      1,MSG(4)
MVI     0(1),NEW_LINE  New Line Char After The Argument
BCTR   4,0
EXRL   4,#+6        Copy Argument To The Message
MVC    MSG(0),0(5)
AHI    4,2
ST     4,MSG_LEN    Save Message Length
CALL   BPX1WRT,MF=(E,SYSCWRT) Print The Message (Argument)
*
LA     2,4(,2)      @Next Argument Length
LA     3,4(,3)      @Next Arugment Data
JCT   7,LOOP
```

(right) click [here](#) to extract the whole program

# Printing Parameters To Standard Output - Compile, Run

\$



here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs
```

here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs  
as -I asma.sasmmac2 -o printargs.o printargs.asm  
Assembler Done No Statements Flagged
```

here



## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs  
as -I asma.sasmmac2 -o printargs.o printargs.asm  
Assembler Done No Statements Flagged  
ld -S "///'sys1.csslib'" -o printargs printargs.o
```

here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs  
as -I asma.sasmmac2 -o printargs.o printargs.asm  
Assembler Done No Statements Flagged  
ld -S "///'sys1.csslib'" -o printargs printargs.o  
$
```

here

# Printing Parameters To Standard Output - Compile, Run

```
$ make printargs  
as -I asma.sasmmac2 -o printargs.o printargs.asm  
Assembler Done No Statements Flagged  
ld -S "///'sys1.csslib'" -o printargs printargs.o  
$ ./printargs abc def ghi
```

here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs
as -I asma.sasmmac2 -o printargs.o printargs.asm
Assembler Done No Statements Flagged
ld -S "///'sys1.csslib'" -o printargs printargs.o
$ ./printargs abc def ghi
0004
```

here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs
as -I asma.sasmmac2 -o printargs.o printargs.asm
Assembler Done No Statements Flagged
ld -S "///'sys1.csslib'" -o printargs printargs.o
$ ./printargs abc def ghi
0004
./printargs
```

here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs
as -I asma.sasmmac2 -o printargs.o printargs.asm
Assembler Done No Statements Flagged
ld -S "///'sys1.csslib'" -o printargs printargs.o
$ ./printargs abc def ghi
0004
./printargs
abc
def
ghi
```

here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs
as -I asma.sasmmac2 -o printargs.o printargs.asm
Assembler Done No Statements Flagged
ld -S "///'sys1.csslib'" -o printargs printargs.o
$ ./printargs abc def ghi
0004
./printargs
abc
def
ghi
$
```

here

## Printing Parameters To Standard Output - Compile, Run

```
$ make printargs
as -I asma.sasmmac2 -o printargs.o printargs.asm
Assembler Done No Statements Flagged
ld -S "///'sys1.csslib'" -o printargs printargs.o
$ ./printargs abc def ghi
0004
./printargs
abc
def
ghi
$
```

(right) click [here](#) to extract a Makefile for compiling and linking the attached programs



# Outline

- 4 The cool stuff
  - Linking to your old code
  - Reading spool from a z/OS UNIX program
  
- 5 Summary
  - Summary
  - Further reading

## Part II

# The cool stuff

## Linking to a STEPLIB

If you want to load or link programs that reside in a dataset:

- Use the DCB= option of the LOAD macro described in [4]
- Set the STEPLIB environment variable:

## Linking to a STEPLIB

If you want to load or link programs that reside in a dataset:

- Use the DCB= option of the LOAD macro described in [4]
- Set the STEPLIB environment variable:

## Linking to a STEPLIB

If you want to load or link programs that reside in a dataset:

- Use the DCB= option of the LOAD macro described in [4]
- Set the STEPLIB environment variable:

## Linking to a STEPLIB

If you want to load or link programs that reside in a dataset:

- Use the DCB= option of the LOAD macro described in [4]
- Set the STEPLIB environment variable:

## Linking to a STEPLIB

If you want to load or link programs that reside in a dataset:

- Use the DCB= option of the LOAD macro described in [4]
- Set the STEPLIB environment variable:

```
$ STEPLIB=dsn1:dsn2:dsn3 command
```

## Linking to a STEPLIB

If you want to load or link programs that reside in a dataset:

- Use the DCB= option of the LOAD macro described in [4]
- Set the STEPLIB environment variable:

```
$ STEPLIB=dsn1:dsn2:dsn3 command
```

```
$ export STEPLIB=dsn1:dsn2:dsn3
```

```
$ command
```

```
$ command
```



## Linking to a STEPLIB

If you want to load or link programs that reside in a dataset:

- Use the DCB= option of the LOAD macro described in [4]
- Set the STEPLIB environment variable:

```
$ STEPLIB=dsn1:dsn2:dsn3 command
```

```
$ export STEPLIB=dsn1:dsn2:dsn3
```

```
$ command
```

```
$ command
```

```
$ STEPLIB=dsn1:dsn2:dsn3
```

```
$ export STEPLIB
```

```
$ command
```

```
$ command
```

# Outline

- 4 The cool stuff
  - Linking to your old code
  - Reading spool from a z/OS UNIX program
  
- 5 Summary
  - Summary
  - Further reading

## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

# Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.



## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

## Reading spool from a z/OS UNIX program

SHARE in Orlando 2008, session 2665, page 40 describes how to use a "spool browse" sample program by Tom Wasik.

[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Orlando/S2665TW195204.pdf](http://proceedings.share.org/client_files/SHARE_in_Orlando/S2665TW195204.pdf)

- This program executes a special form of SVC 99 that allows reading from the spool several files related to a JOB:
  - JCL - Originally submitted JCL
  - JESJCL - Effective JCL as processed by JES
  - JESMSGLG - messages
  - JESYSMSG - allocation messages
- Open the pdf in Adobe Acrobat Reader, click on the "Attachments" icon, then right click the "BROWSE.txt" attachment, and select "Save attachment".

As an exercise make this program run under z/OS UNIX.

## Reading spool from a z/OS UNIX program, cont'd

Using the exercise from previous slide you can for example:

- Find all the the steps in a JCL
- Find all the the steps in a JCL after expanding a PROC
- And many more using z/OS UNIX text processing utilities and languages (e.g. grep, sed, awk, perl)

## Reading spool from a z/OS UNIX program, cont'd

Using the exercise from previous slide you can for example:

- Find all the the steps in a JCL
- Find all the the steps in a JCL after expanding a PROC
- And many more using z/OS UNIX text processing utilities and languages (e.g. grep, sed, awk, perl)

## Reading spool from a z/OS UNIX program, cont'd

Using the exercise from previous slide you can for example:

- Find all the the steps in a JCL
- Find all the the steps in a JCL after expanding a PROC
- And many more using z/OS UNIX text processing utilities and languages (e.g. grep, sed, awk, perl)

## Reading spool from a z/OS UNIX program, cont'd

Using the exercise from previous slide you can for example:

- Find all the the steps in a JCL

```
$ ./browse GOTVI01.Y8VSMPL.JOB58835.JCL | grep EXEC
//ASMCLG EXEC HLASMCL
//RUNPGM EXEC PGM=GO,PARM='HELLO'
$
```

- Find all the the steps in a JCL after expanding a PROC

- And many more using z/OS UNIX text processing utilities and languages (e.g. grep, sed, awk, perl)

## Reading spool from a z/OS UNIX program, cont'd

Using the exercise from previous slide you can for example:

- Find all the the steps in a JCL

```
$ ./browse GOTVI01.Y8VSMPL.JOB58835.JCL | grep EXEC
//ASMCLG EXEC HLASMCL
//RUNPGM EXEC PGM=GO,PARM='HELLO'
$
```

- Find all the the steps in a JCL after expanding a PROC

- And many more using z/OS UNIX text processing utilities and languages (e.g. grep, sed, awk, perl)



## Reading spool from a z/OS UNIX program, cont'd

Using the exercise from previous slide you can for example:

- Find all the the steps in a JCL

```
$ ./browse GOTVI01.Y8VSMPL.JOB58835.JCL | grep EXEC
//ASMCLG EXEC HLASMCL
//RUNPGM EXEC PGM=GO,PARM='HELLO'
$
```

- Find all the the steps in a JCL after expanding a PROC

```
$ ./browse GOTVI01.Y8VSMPL.JOB58835.JESJCL | grep EXEC
 2 //ASMCLG EXEC HLASMCL
 4 XXC EXEC PGM=ASMA90,PARM=(OBJECT,NODECK),REGION=1M
14 XXL EXEC PGM=HEWL,PARM='MAP,LET,LIST,NCAL',COND=(8,LT,C)
20 //RUNPGM EXEC PGM=GO,PARM='HELLO'
$
```

- And many more using z/OS UNIX text processing utilities and languages (e.g. grep, sed, awk, perl)

## Reading spool from a z/OS UNIX program, cont'd

Using the exercise from previous slide you can for example:

- Find all the the steps in a JCL

```
$ ./browse GOTVI01.Y8VSMPL.JOB58835.JCL | grep EXEC
//ASMCLG EXEC HLASMCL
//RUNPGM EXEC PGM=GO,PARM='HELLO'
$
```

- Find all the the steps in a JCL after expanding a PROC

```
$ ./browse GOTVI01.Y8VSMPL.JOB58835.JESJCL | grep EXEC
 2 //ASMCLG EXEC HLASMCL
 4 XXC EXEC PGM=ASMA90,PARM=(OBJECT,NODECK),REGION=1M
14 XXL EXEC PGM=HEWL,PARM='MAP,LET,LIST,NCAL',COND=(8,LT,C)
20 //RUNPGM EXEC PGM=GO,PARM='HELLO'
$
```

- And many more using z/OS UNIX text processing utilities and languages (e.g. grep, sed, awk, perl)

## Couple more goodies

- Once a program can run under z/OS UNIX, you can use it in shell scripts and combine it with other z/OS UNIX commands.
- As a side effect you can also call it from z/OS UNIX REXX execs.
- You can use SSH to execute the program remotely from your Windows or Linux PC and get the output directly on the PC and further process it.  
Ed Jaffe presented on this at SHARE in Anaheim in 2011,  
[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Anaheim\\_2/Session\\_8666\\_handout\\_1051\\_0.pdf](http://proceedings.share.org/client_files/SHARE_in_Anaheim_2/Session_8666_handout_1051_0.pdf)
- Your next new hire is likely to know UNIX better than MVS and you could make him/her productive and creative day one.

## Couple more goodies

- Once a program can run under z/OS UNIX, you can use it in shell scripts and combine it with other z/OS UNIX commands.
- As a side effect you can also call it from z/OS UNIX REXX execs.
- You can use SSH to execute the program remotely from your Windows or Linux PC and get the output directly on the PC and further process it.  
Ed Jaffe presented on this at SHARE in Anaheim in 2011,  
[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Anaheim\\_2/Session\\_8666\\_handout\\_1051\\_0.pdf](http://proceedings.share.org/client_files/SHARE_in_Anaheim_2/Session_8666_handout_1051_0.pdf)
- Your next new hire is likely to know UNIX better than MVS and you could make him/her productive and creative day one.

## Couple more goodies

- Once a program can run under z/OS UNIX, you can use it in shell scripts and combine it with other z/OS UNIX commands.
- As a side effect you can also call it from z/OS UNIX REXX execs.
- You can use SSH to execute the program remotely from your Windows or Linux PC and get the output directly on the PC and further process it.  
Ed Jaffe presented on this at SHARE in Anaheim in 2011,  
[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Anaheim\\_2/Session\\_8666\\_handout\\_1051\\_0.pdf](http://proceedings.share.org/client_files/SHARE_in_Anaheim_2/Session_8666_handout_1051_0.pdf)
- Your next new hire is likely to know UNIX better than MVS and you could make him/her productive and creative day one.

## Couple more goodies

- Once a program can run under z/OS UNIX, you can use it in shell scripts and combine it with other z/OS UNIX commands.
- As a side effect you can also call it from z/OS UNIX REXX execs.
- You can use SSH to execute the program remotely from your Windows or Linux PC and get the output directly on the PC and further process it.

Ed Jaffe presented on this at SHARE in Anaheim in 2011,  
[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Anaheim\\_2/Session\\_8666\\_handout\\_1051\\_0.pdf](http://proceedings.share.org/client_files/SHARE_in_Anaheim_2/Session_8666_handout_1051_0.pdf)

- Your next new hire is likely to know UNIX better than MVS and you could make him/her productive and creative day one.

## Couple more goodies

- Once a program can run under z/OS UNIX, you can use it in shell scripts and combine it with other z/OS UNIX commands.
- As a side effect you can also call it from z/OS UNIX REXX execs.
- You can use SSH to execute the program remotely from your Windows or Linux PC and get the output directly on the PC and further process it.  
Ed Jaffe presented on this at SHARE in Anaheim in 2011,  
[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Anaheim\\_2/Session\\_8666\\_handout\\_1051\\_0.pdf](http://proceedings.share.org/client_files/SHARE_in_Anaheim_2/Session_8666_handout_1051_0.pdf)
- Your next new hire is likely to know UNIX better than MVS and you could make him/her productive and creative day one.

## Couple more goodies

- Once a program can run under z/OS UNIX, you can use it in shell scripts and combine it with other z/OS UNIX commands.
- As a side effect you can also call it from z/OS UNIX REXX execs.
- You can use SSH to execute the program remotely from your Windows or Linux PC and get the output directly on the PC and further process it.  
Ed Jaffe presented on this at SHARE in Anaheim in 2011,  
[http://proceedings.share.org/client\\_files/SHARE\\_in\\_Anaheim\\_2/Session\\_8666\\_handout\\_1051\\_0.pdf](http://proceedings.share.org/client_files/SHARE_in_Anaheim_2/Session_8666_handout_1051_0.pdf)
- Your next new hire is likely to know UNIX better than MVS and you could make him/her productive and creative day one.



# Outline

- 4 The cool stuff
  - Linking to your old code
  - Reading spool from a z/OS UNIX program
- 5 Summary
  - Summary
  - Further reading

# Summary

- A program running in z/OS belongs to an address space and executes under a TCB - whether it is regular MVS or z/OS UNIX program.
- You can use both MVS and z/OS UNIX services whether running under batch, TSO, or z/OS UNIX.
- If you have an existing MVS application, you can make it accessible from z/OS UNIX just by creating a front end that will:
  - Parse arguments passed from the shell
  - Pass them to your application in the usual way
  - Printing the output of your application to the STDOUT

# Outline

- 4 The cool stuff
  - Linking to your old code
  - Reading spool from a z/OS UNIX program
  
- 5 Summary
  - Summary
  - Further reading

# For Further Reading I

- [1] z/OS UNIX System Services Programming:  
*Assembler Callable Services Reference*, [▶ SA22-7803](#)
- [2] z/OS UNIX System Services:  
*Messages and Codes*, [▶ SA22-7807](#)
- [3] ABCs of z/OS System Programming:  
*Volume 9*, [▶ SG24-6989](#)
- [4] z/OS MVS Programming:  
*Assembler Services Guide*, [▶ SA22-7605](#)
- [5] z/OS MVS Programming:  
*Authorized Assembler Services Guide*, [▶ SA22-7608](#)

# For Further Reading II

- [6] Steve Comstock  
*z/OS, Language Environment, and UNIX*  
*The Trainer's Friend, Inc., 2012,* [▶ z/OS UNIX and LE](#)

Please, do not forget to fill in evaluation

