



# Finding Gold with the z/OS UNIX APIs

Clark Kidd

Session 2913; SHARE 105 in Boston, MA

August 23, 2005

Watson & Walker, Inc. / [www.watsonwalker.com](http://www.watsonwalker.com)

Home of "Cheryl Watson's TUNING Letter", BoxScore and GoalTender

# Agenda for This Presentation

- Introduction
- Assembler Language Considerations
- REXX Language Considerations
- C/C++ Language Considerations
- Some of the More Useful APIs
- Other Considerations
- References

# Introduction



- About This Presentation
  - Help You Avoid “Rube Goldberg” Solutions
  - An “Iceberg” Presentation (90% Hidden)
  - Very Technical / Lots of Sample Code

# Introduction

- z/OS Program Scope
  - MVS World Only
  - UNIX World Only
  - Both MVS and UNIX Worlds (Mixed-Mode programs)
  - There is no Wall Between Worlds!
- Living in Both Worlds
  - MVS Operator Commands (D OMVS)
  - JCL Options (PATH, PATHMODE)
  - TSO Commands (OGET, OPUT)
  - OMVS Shell Commands
  - Program APIs (Assembler, REXX, C/C++)

# Introduction

- General Categories of API Functions
  - Application Management (Process/Thread)
  - File and Directory Management
  - IPC Management (Interprocess Communication)
  - Security Management
  - Network and Socket Management
  - System Management
- There are 240+ Assembler Language UNIX APIs!

# Assembler Language Considerations

- General
  - Program Must be AMODE=31
  - Invoke API via CALL Macro
    - R1 = Address of Parameter List
    - R14 = Return Address of Calling Program
    - R15 = Entry Point Address of API
    - Use VL Option on Macro
  - API Names are in the Format BPX1xxx
  - Some APIs use the Format BPX2xxx/BPX4xxx
    - Standard versus Non-Standard Implementation

# Assembler Language Considerations



- Getting the API Address (Method 1)

LOAD	EP=BPX1GPI	LOAD	SERVICE
LR	R15,R0	COPY	ADDRESS
CALL	(15),	CALL	THIS ADDRESS
	(RESULT),	PUT	RESULT HERE
VL		INDICATE	LAST PARAMETER
DELETE	EP=BPX1GPI	DELETE	THE SERVICE

- Getting the API Address (Method 2)

\* REMEMBER TO INCLUDE SYS1.CSSLIB IN LINKAGE STEP

L	R15,=V(BPX1GPP)	ADDRESS	OF LINKAGE STUB
CALL	(15),	CALL	THIS ADDRESS
	(RESULT),	PUT	RESULT HERE
VL		INDICATE	LAST PARAMETER





# Assembler Language Considerations



- Parameters of the CALL Macro
  - Parameter-1 - Parameter-n
    - Number Varies Depending on the Service
    - Specify Input and Output Values
    - Mapped by Various BPXY\* Macros
  - RETURN VALUE (Fullword)
    - Used by Services that Can Fail
    - -1 Usually Indicates an Error (Sometimes 0)
  - RETURN CODE (Fullword)
    - Error Return Code when a Service Fails
    - Called “errno” in some UNIX Documentation
    - Values Mapped by BPXYERNO Macro

# Assembler Language Considerations



- Parameters of the CALL Macro (Continued)
  - REASON CODE (Fullword)
    - Reason Code when a Service Fails
    - Format MMMRRRRR (M=Module ID, R=Reason Code)
    - Not Part of the POSIX Standard (z/OS UNIX only)
    - Called “errnojr” in some IBM UNIX Documentation
    - Values Mapped by BPXYERNO Macro

ICM R1,15,VALUE

BNM CONTINUE

LA R1,ESRCH

C R1,RETURN

BNE UNHANDLE

Q. SERVICE WORK?

A. YES, THAT WAS GOOD

LOAD RETURN CODE

Q. PID NOT FOUND?

A. YES, HANDLE THAT

\* HANDLE CONDITION WHERE THE REQUESTED PID WAS NOT FOUND

BPXYERNO ,

MAP RETURN CODES

# REXX Language Considerations



- REXX with UNIX Syscalls May be Run:
  - Under TSO/E
  - Under the z/OS UNIX Shell
  - In Batch
    - IKJEFT01
    - IRXJCL
    - BPXBATCH
  - From Another Program
- Consider Platform when Writing Code
  - Some Subtle Differences

# REXX Language Considerations



- For TSO and Batch REXX Initialization:
  - `call syscalls 'ON'`
  - `call syscalls 'OFF'`
  - Seems to be Optional in Some Cases
- Coding for Multiple Platforms (TSO and OMVS)

```
/* REXX */  
parse source . . . . . envt .  
if envt <> "OMVS" then  
    call syscalls 'ON'
```

# REXX Language Considerations



- Calling the APIs
  - Include `address syscall` Statement
  - Include `'API name in quotes'`
  - Use Other `address` Statements as Needed

- Example

```
/* REXX */  
call syscalls 'ON'  
address syscall  
'getpid'  
say 'Your PID is' retval  
address tso  
say 'Your Data Sets:'  
'listc'
```

# REXX Language Considerations



- Complex Results Returned in Stem Variables

```
'statfs (filenm) fs.'
```

```
say filenm
```

```
say ' Total Blocks = ' fs.stfs_total
```

```
say ' Free Blocks = ' fs.stfs_bfree
```

- Sample Output

```
HFS.WEBDATA
```

```
Total Blocks = 2160
```

```
Free Blocks = 1859
```

- API Description Includes Stem Definitions
- REXX is a Good Tool for Prototypes

# REXX Language Considerations



- Detecting & Reporting Errors / Do It Yourself
  - Variables RETVAL, ERRNO, ERRNOJR

- Example

```
'statfs (filenm) fs.'  
if retval = -1 then do  
    say 'Error processing File System:' filenm  
    say 'Return=' errno '/ Reason=' errnoj  
end
```

- Output

```
Error processing File System: HFS.WEBLIB2  
Return= 79 / Reason= 567002E
```

# REXX Language Considerations



- Detecting & Reporting Errors / STRError
- Example

```
if retval = -1 then do
  'strerror' errno errnojrc 'erd.'
  say 'Error   =' erd.se_errno
  say 'Reason =' erd.se_reason
  say 'Module =' erd.se_modid
end
```

- Output

```
Error   = EINVAL: The parameter is incorrect
Reason  = JRFilesysNotThere: The file system named
        does not exist
Module  = BPXFSSTF09/17/01
```



# C/C++ Language Considerations



- General
  - No Initialization Needed
  - Just Use Desired UNIX Library Functions
  - Some Require **POSIX (ON)** Run-Time Option
- To Use the Functions
  - Include Header Files as Needed
  - Define Structures and Storage as Needed
  - Code the Function

```
rc = function_name (parm1 , parm2...) ;
```
  - Test Return Code and Process Results

# C/C++ Language Considerations



- Coding Example - See if a File Exists

```
#define _POSIX_SOURCE
#include <unistd.h>
#include <stdio.h>
main() {
char path[] = "/usr/include/ctype.h";
printf("File %s ", path);
if (access(path, F_OK) != 0)
    printf("does NOT exist!\n");
else printf("exists.\n");
}
```

# C/C++ Language Considerations



- Detecting & Reporting Errors / Do It Yourself
  - Include `<errno.h>` Header File
    - Defines `errno` as a Numeric Variable
    - Defines Constant Values for `errno`

- Example

```
#include <errno.h>
main() {
    struct statvfs buf;
    char *myfile = "/usr/include/missing.h";
    if (statvfs(myfile, &buf) == -1) {
        if (errno == ENOENT)
            printf("Specified File Does Not Exist.\n");
        else printf("Some other error?"); }}
```

# C/C++ Language Considerations



- Detecting & Reporting Errors / PERROR
- Example

```
#include <stdlib.h>
main() {
    struct statvfs buf;
    setenv("_EDC_ADD_ERRNO2", "1", 1);
    char *myfile = "/usr/include/missing.h";
    if (statvfs(myfile, &buf) == -1)
        perror("We had an error"); }
```

- Output

```
We had an error: EDC5129I No such file
or directory. (errno2=0x05F1006C)
```

# C/C++ Language Considerations



- Detecting & Reporting Errors / STRError
- Example

```
#include <string.h>
#include <stdlib.h>
#include <errno.h>
main() {
    struct statvfs buf;
    setenv("_EDC_ADD_ERRNO2", "1", 1);
    char *myfile = "/usr/include/missing.h";
    if (statvfs(myfile, &buf) == -1)
        printf("Error: %s\n", strerror(errno)); }
```

- Output

```
Error: EDC5129I No such file
or directory. (errno2=0x05F1006C)
```

# Some of the More Useful APIs

## getpsent, w\_getpsent(), BPX1GPS



- Function
  - Return Information about Active UNIX Processes
- Input
  - Process Token; Address of the Output Area
- Output
  - Process Identifiers (PID, GID, UID...)
  - Configuration Data
  - CPU Consumption Data
  - Mapped by
    - `ps` stem / `w_PSPROC` structure / `BPXYGPS` Macro

# Some of the More Useful APIs

## getpsent, w\_getpsent(), BPX1GPS



- REXX Example

```
'getpsent ps.'  
do i=1 to ps.0  
    say right(ps.i.ps_pid,10) ,  
        right(ps.i.ps_pgpid,10) ,  
        right(ps.i.ps_size,10) ,  
        ' ' ps.i.ps_cmd  
end
```

- Example Output

PID	GID	Size	Command
1	1	114688	BPXPINPR
65538	65538	67448832	IMWHTTPD
33619980	33619980	2109440	GFSCRPCD

# Some of the More Useful APIs

## `getmntent`, `w_getmntent()`, `BPX1GMN`



- Function
  - Return List of Mounted File Systems
- Input
  - Address of the Output Area; Optional Device ID
- Output
  - File System Status
  - Configuration Data
  - I/O Counts
  - Mapped by
    - `mntent` stem / `mntent.h` header / `BPXYMNTTE` Macro



# Some of the More Useful APIs

## getmntent, w\_getmntent(), BPX1GMN



- REXX Example

```
'getmntent mt.'  
do i=1 to mt.0  
  say mt.mnte_fsname.i  
  say '  Type =' mt.mnte_fstype.i,  
    'Reads =' mt.mnte_readct.i,  
    'Writes =' mt.mnte_writect.i,  
    'Path =' mt.mnte_path.i  
end
```

- Example Output

```
HFS.PROD.ETC
```

```
  Type = HFS Reads = 23 Writes = 101 Path = /etc
```

```
HFS.TESTPX.ROOT
```

```
  Type = HFS Reads = 70 Writes = 0 Path = /
```

## Some of the More Useful APIs

**statfs/statvfs, statvfs(), BPX1STF/BPX1STV**



- Function
  - Return Information about a Mounted File System
- Input
  - Address of the Output Area
  - File System Name or File Name/Path
- Output
  - File System Status and Configuration Data
  - Space Information
  - Mapped by
    - **stfs** stem / **statvfs** struct / **BPXYSSTF** Macro

# Some of the More Useful APIs

## statfs/statvfs, statvfs(), BPX1STF/BPX1STV



- C++ Example

```
printf("Data For Root File System\n");  
if (statvfs("/", &buf) == -1) printf("Error\n");  
else {  
    printf("File System ID = %d\n",buf.f_fsid);  
    printf("Alloc Blocks    = %d\n",buf.f_OEusedspace);  
    printf("Free Blocks      = %d\n",buf.f_bfree); }  
}
```

- Example Output

```
Data For Root File System  
File System ID = 1  
Alloc Blocks    = 20  
Free Blocks     = 160
```

# Some of the More Useful APIs

## stat, stat(), BPX1STA



- Function
  - Returns Information about a UNIX File
- Input
  - Full or Relative UNIX File Name
- Output
  - Times (Created, Accessed, Changed)
  - Characteristics (Sizes and Formats)
  - Owners (UID and GID)
  - Mapped by
    - `stat` stem / `stat` structure / `BPXYSTAT` Macro

# Some of the More Useful APIs

## stat, stat(), BPX1STA



- C++ Example

```
struct stat info;
if (stat(argv[1], &info) != 0) perror("error");
else {
    printf("Statistics for: %s.\n", argv[1]);
    printf(" Inode:          %d\n", (int) info.st_ino);
    printf(" Device ID: %d\n", (int) info.st_dev);
    printf(" Mode:          %08x\n", info.st_mode); }
}
```

- Example Output

```
Statistics for: /usr/include/ctype.h.
Inode:          39341
Device ID: 3
Mode:          030001a4
```

# Some of the More Useful APIs

## getpwent, getpwent(), BPX1GPE



- Function
  - Sequential Access to User Security Data Base
- Input
  - None / Call Until All Entries are Returned
- Output
  - User Name
  - User ID (UID) and Group ID (GID)
  - Initial Shell Command and Directory
  - Mapped by
    - **passwd** stem / **passwd** structure / **BPXYGIDN** Macro

# Some of the More Useful APIs

## getpwent, getpwent(), BPX1GPE



- REXX Example

```
do forever
  'getpwent pw.'
  if retval=0 | retval=-1 then
    leave
  say 'Name=' pw.pw_name 'UID=' pw.pw_uid ,
      'GID=' pw.pw_gid
end
```

- Example Output

```
Name= IRVING      UID= 537  GID= 500
Name= PAUL        UID= 416  GID= 255
Name= PETER       UID= 998  GID= 999
Name= WALT        UID= 206  GID= 205
```

## Some of the More Useful APIs getgrent, getgrent(), BPX1GGE



- Function
  - Sequential Access to Group Security Data Base
- Input
  - None / Call Until All Entries are Returned
- Output
  - Group Name and Group ID (GID)
  - Number and Names of Users in Group
  - Mapped by
    - **group** stem / **group** structure / **BPXYGIDS** Macro



# Some of the More Useful APIs

## getgrent, getgrent(), BPX1GGE



- REXX Example

```
do forever
  'getgrent gr.'
  if retval=0 | retval=-1 then
    leave
  say 'Name=' gr.gr_name 'GID=' gr.gr_gid ,
    'Number=' gr.gr_members
end
```

- Example Output

```
Name= WEBSPHR   GID= 205 Number= 2
Name= OPERGRP   GID= 1 Number= 2
Name= PAYROLL   GID= 255 Number= 1
Name= SYSPROG   GID= 0 Number= 15
```

# Some of the More Useful APIs

## uname, uname(), BPX1UNA



- Function
  - Return Host Machine Configuration Data
- Input
  - Address of the Output Area
- Output
  - Hardware Machine Type
  - Operating System Type, Release and Version
  - Host System ID
  - Mapped by
    - `utsname` stem / `utsname` structure / `BPXYUTSN` Macro

# Some of the More Useful APIs

## uname, uname(), BPX1UNA



- REXX Example

```
'uname us.'  
say 'Hardware Type =' us.u_machine  
say 'System ID     =' us.u_nodename  
say 'OS Type       =' us.u_sysname  
say 'OS Release    =' us.u_release  
say 'OS Version    =' us.u_version
```

- Example Output

```
Hardware Type = 1247  
System ID     = TSTA  
OS Type       = OS/390  
OS Release    = 14.00  
OS Version    = 03
```

# Some of the More Useful APIs

## n/a, n/a, BPX1RMG



- Function
  - Return Resource Measurement Data
- Input
  - Address of the Statistics Area
- Output
  - Total System Calls and UNIX CPU Usage
  - Threshold Values
    - Maximum and Current
    - Number of Attempts to Exceed Maximum
  - Mapped by
    - **BPXYRMON** Macro

# Some of the More Useful APIs

## n/a, n/a, BPX1RMG



- Assembler Example

L	R15,=V(BPX1RMG)	LINKAGE STUB
CALL	(15),	CALL THIS ADDRESS
	(RMONLEN,	LENGTH OF RESULT
	RMONADR,	RESULT AREA
	RETVAL,RC,REASON),VL	RETURN VALUES

- Example Output

TOTAL SYSCALLS ----->	253112
CURRENT PROCESS COUNT -->	24
MAXIMUM PROCESS COUNT -->	200
ATTEMPTS TO EXCEED ----->	0

# Some of the More Useful APIs

## n/a, n/a, BPXESMF Macro



- Function
  - Return UNIX SMF Statistics for Address Space
  - Output Similar to SMF-30 OpenMVS Section
- Input
  - Address of ASCB for Address Space
- Output
  - Identifiers, SYSCALL Count and CPU Usage
  - I/O Counts and Service Counts
  - Mapped by
    - **BPXYOSMF** Macro

# Some of the More Useful APIs

## n/a, n/a, BPXESMF Macro



- Assembler Example

```
MODESET MODE=SUP,KEY=ZERO          GET AUTHORIZED
BPXESMF ACCTDAT=TESTOSMF,          A (OUTPUT AREA)
      ASCBPTR= (R5)                 A (INPUT ASCB)
MODESET MODE=PROB,KEY=NZERO        GET SAFE AGAIN
LA      R6,TESTOSMF                 A (RETURNED AREA)
USING  OSMF,R6                      MAP IT
```

- Example Output

JOBNAME	SYSCALLS	DIR--READS
-----	-----	-----
TCPIP	992	34
HTTPD1	394018	2387
BPXOINIT	26	0

## Some of the More Useful APIs

n/a, `_get_system_settings()`, BPXEKDA Macro



- Function
  - Obtain UNIX Kernel Information
  - Program Access to “D O M V S” Command Data
- Input
  - Address/Length/ALET of Input/Output Buffer
- Output
  - Many Different Output Options
  - System Level or Process Level
  - Mapped by
    - `_optn` Structure / BPXZODMV Macro



## Some of the More Useful APIs n/a, \_get\_system\_settings(), BPXEKDA Macro



- Assembler Example

```
MODESET MODE=SUP,KEY=ZERO          AUTHORIZATION
BPXEKDA KBUFLN=TESTBUFL,          BUFFER LENGTH
      KBUFALET=TESTALET,          ALET = 0
      KBUFPTR=TESTBUFA           ADDRESS OF BUFFER
LR      R3,R15                    SAVE RETURN CODE
MODESET MODE=PROB,KEY=NZERO        PROBLEM MODE
```

- Example Output

```
OMVS PROC NAME -----> OMVS
BPXPRMXX IN EFFECT -----> OMVS=(CS)
MAXIMUM PROCESSES -----> 200
MAXIMUM USERS -----> 200
```

# Some of the More Useful APIs

## BPXWDYN



- A Text (String) Interface to
  - Dynamic Allocation (SVC 99)
  - Dynamic Output (SVC 109)
- Supported Functions
  - Allocate/Unallocate/Concatenate Data Sets
  - Add/Delete Output Descriptors
- Called from
  - REXX
  - Assembler Language; C/C++; PL/I
  - TSO or non-TSO Environments
- See REXX Manual for Details

## Other Considerations

- Output from API May Vary
  - Based on Your Operating System Release
  - Based on Your Security Profile
- API Input Parameter Lists Vary
  - Some Values are Addresses of Objects
  - Some Values are Addresses of Addresses
- Return Values from API Vary
  - Sometimes Zero is Okay
  - Sometimes Zero is an Error

## Other Considerations

- Multiple Variations of Similar Services
  - Access Data Base Sequentially (**getgrent**)
  - Access Data Base by ID (**getgrgid**)
  - Access Data Base by Name (**getgrnam**)
- Slight API Variations Based on Platform
  - Assembler Version Usually Richest in Features
- Bottom Line: Read Documentation Carefully
- Combine APIs for Maximum Power
  - Get List of Mounted File Systems
  - Get Space Usage for each File System

## References

- UNIX System Services Programming: Assembler Callable Services Reference (SA22-7803)
- Using REXX and z/OS UNIX System Services (SA22-7806)
- C/C++ Run-Time Library Reference (SA22-7821)
- BPXESMF and BPXEKDA Macros: Authorized Assembler Services Reference; Volume 1 (SA22-7609)
- *Unveiling the Secrets of USS Error Messages*; Cheryl Watson's TUNING Letter; 2002, No. 4, p. 23.

## References

- Bit Bucket X'1A'; Bob Shannon; Session 2817; SHARE 101 (Washington D.C.); August, 2003
- UNIX Tools and Toys: <http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1toy.html>
- UNIX “**configfs**” command source (written in REXX): `/usr/lpp/dfsms/bin/configfs`
- My Sample Programs:
  - [www.watsonwalker.com/presentations.html](http://www.watsonwalker.com/presentations.html)
- Questions?

[clark@watsonwalker.com](mailto:clark@watsonwalker.com)

[www.watsonwalker.com](http://www.watsonwalker.com)