



CMS Application Debugging

Phil Smith III





Why We're Here

- We all run OPP (Other People's Products)
- Programs (even CA's!) *do* fail
- You do not always have source code or formal diagnostic procedures available

How do you approach and resolve problems most effectively?



Agenda

- Define problem types
- Discuss tools
- Examine some sample scenarios



Disclaimer

- This presentation is partly based on recent experiences supporting CA products
 - ✓ Information provided is *not* CA-specific
 - ✓ Use it to improve both local and vendor problem diagnosis/resolution
- **Not** an attempt to teach dump reading
 - ✓ (Sadly) dump reading is a dying art
 - ✓ Typically not learned without hands-on anyway



My Background

- 20+ years of VM development and support
- I currently support 16 or so products
 - ✓ Many of these I've never used in the real world
 - ✓ CS reps are often surprised that I can fix things without even knowing how to test them
- How?
 - ✓ I know and use the tools I have available
 - ✓ Use different tools/techniques until one works



Debugging 101

- Generalizing debugging heuristics is hard!
 - ✓ Many cases come down to “Look at stuff, see what’s weird” (notice any anomalies)
- If you don’t understand something, ***don’t*** just ignore it – it may be the key
 - ✓ Examples: compiler warnings from PL/I, C
 - ✓ The more clues you can collect, the better!

Problem Types



Problem Types

- Most (all?) problems can be grouped as one of the following:
 - ✓ ABEND
 - ✓ Error message
 - ✓ Loop
 - ✓ Incorrect output (“Incorroust”)
- Incorroust is the catch-all
 - ✓ Others are all arguably Incorroust as well!



Problem Subtypes

- Problems may be:
 - ✓ Repeatable (occurs every time)
 - ✓ Non-repeatable (only occurs once/sometimes)
- And either type may be:
 - ✓ Portable (happens on any user, any system)
 - ✓ Non-portable (occurs only on specific userid or in specific environment)
- Always try to characterize
- Non-repeatable, non-portable incorrout are the worst to diagnose

Debugging Tools

COMPUTER[®]
ASSOCIATES



Basics – Spooling the Console

- Spooled console
 - ✓ ***Always*** run with console spooled (and started!) to yourself
 - ✓ Always save console which ***matches*** dump, etc.
- “Real programmers always spool their consoles!”



CMS Tools – Basics

■ LISTFILE

- ✓ Don't laugh – many problems caused by outdated/duplicate EXECs/MODULEs!

■ MODMAP

- ✓ Module information, including entry points
- ✓ IBM version semi-useless; improved **MODMAP** on 1998 VM Workshop site
- ✓ Sorts EPs, shows options, etc.
- ✓ IBM has code, will (hopefully) add to CMS



CMS Tools – Basics

■ NUCXMAP, PROGMAP, EXECMAP, RTNMAP

- ✓ Show nucleus extensions (NUCEXTs), loaded programs, in-storage EXECs, CSL routines

■ NUCXDROP, EXECDROP, RTNDROP

- ✓ Remove NUCEXTs/EXECs from storage
- ✓ Dropping them often helps reduce indeterminacy (“Which one are we running?”)
- ✓ Beware **PERM** nucleus extensions: **NUCXDROP *** does not remove, must drop explicitly



CMS Tools – Basics

- **SEARCH4** or equivalent (source scanner)
 - ✓ Invaluable for finding message texts, program calls from EXECs, function calls, etc.
 - ✓ Aids quick identification of problem module
- **SET EXEC TRAC ON**
 - ✓ Starts EXEC 2 and Rexx tracing
 - ✓ Study program flow, examine variables, etc.
 - ✓ '01' bit at x'5E6' turns on tracing (do **D S5E6**, add 1, then **STORE S5E6 xx**)



CMS Tools – Basics

- **XCOMPARE** or equivalent: comparison tool
 - ✓ Compare before/after, working/non-working (console, **STORMAP** output, control blocks, etc.)
- Much smarter than native **COMPARE**:
 - ✓ Can resynchronize after added/deleted lines
 - ✓ Option to create CMS update files as output
 - ✓ Other options aid comparison “smarts”
- Also **DIFF**, **MATCH**, other names ...



CMS Tools – Basics

■ IPL CMS

- ✓ Program checks, storage allocations, etc.
should occur at same location after each IPL
- ✓ Allows repeatable traces, etc.

■ Consider **ACCESS (NOPROF** to avoid all automatic **ACCESSES**

■ **IPL CMS PARM NOSPROF** skips **SYSPROF EXEC** execution

■ Avoid stacking commands during **PROFILE** – stack uses storage, changes results!



CMS Tools – Message-related

■ CP SET EMSG ON

- ✓ Adds headers on error messages
- ✓ Real Programmers always run with **EMSG ON!**

■ SET TRAPMSG

- ✓ CMS facility to take automatic VMDUMP
- ✓ Only works with error messages (those affected by CP **SET EMSG**)
- ✓ Limited utility, but occasionally useful



CMS Tools – Environmental Aids

■ SET AUTODUMP ALL ENTIREVM

- ✓ Tells CMS to automatically take VMDUMP on CMS or program ABEND
- ✓ Odd syntax: **ALL** means “all ABENDs”, **ENTIREVM** means “dump all storage”

■ EXECOS

- ✓ Call before/after command from EXEC to reset OS Simulation (and VSAM) environments
- ✓ Not needed from console – CMS does automatically



CMS Tools – Environmental Aids

■ CP SET 370ACCOM ON

- ✓ In XA/ESA/XC-mode virtual machine, allows most 370-only programs to run
- ✓ Some limitations, but always worth a try

■ CMS SET CMS370AC ON

- ✓ Traps a few more things at CMS level which 370ACCOM does not
- ✓ Not needed for most programs; try 370ACCOM alone first



CMS Tools – Environmental Aids

■ SET RELPAGE OFF

- ✓ Avoids CMS storage release to CP
- ✓ May change behavior of problem
- ✓ Should ***not*** be considered a fix, but may aid in understanding storage-related bugs

■ SET STORECLR ENDCMD

- ✓ Releases program storage at return to CMS command level, rather than after each SVC
- ✓ Again, may ***change*** behavior – ***not*** a fix



CMS Tools – Storage-related

■ STORMAP

- ✓ Shows allocated/free storage queues
- ✓ Issue before and after command to see storage consumption (cancer)
- ✓ Consider “first run” effects: run program with **STORMAP** three times, compare *last two* runs

■ SUBPMAP

- ✓ Lists storage subpools, allocations for each
- ✓ Allows easy identification of heavy storage use



CMS Tools – Storage-related

- **HX** after command, to clear storage
 - ✓ Releases **USER** storage subpool
 - ✓ Combined with **STORMAP / SUBPMAP**, can help identify storage cancers
- **STDEBUG**
 - ✓ CMS storage allocation/deallocation trap
 - ✓ Options include ranges, subpools, tracing to punch or via **CP MSG** to other users



CMS Tools – Storage-related

24-bit vs. 31-bit issues:

- Try to reproduce errors with 15MB and with 32MB (or more): may only occur with $> 16\text{MB}$
- Especially for PROG1, PROG5 ABENDs
- With 15MB, even 31-bit stuff is below-the-line
- PROG4 in 24-bit may change to PROG5 in 31-bit (due to storage “wraparound”)



CMS Tools – SVCTRACE

- CMS **SVCTRACE** facility traces registers before and after SVC calls
- Indicates if SVC ended with non-zero RC
- Shows SVC depth, caller, callee, arguments
- **so/HO CMS Immediate commands start/end SVCTRACE**
- Output is to virtual printer
- Of limited utility, occasionally useful



CP Tools – TRACE

- **TRACE** in VM/ESA is synonym for **PER**
- **TRACE PROG** stops on program checks
 - ✓ Some program checks are normal (for example, during CMS IPL)
- **TRACE BRANCH** traps branches from/into address ranges
 - ✓ High overhead, sometimes impractical
 - ✓ When trap occurs, **TRACE TABLE** shows last six branches trapped – often useful



CP Tools – TRACE

- **TRACE STORE** traps *most* storage changes
 - ✓ Can trace alterations to registers or specific locations, including specific values
 - ✓ Causes significant execution slowdown
 - ✓ Does not trap alterations by I/O, DIAGNOSE, MVCL, some other opcodes unless specific value specified
 - ✓ Hardware limitation, not easily overcome



CP Tools – TRACE

- **TRACE CALL/RETURN/GOTO** allow easy movement among multiple trace sets
- **QUERY TRACE, QUERY TRACE SETS, QUERY TRACE RETURNS** display traps, named sets, **TRACE CALL** structure
- **TRACE CMD** adds automatic action on trap
 - ✓ Consider chaining traps together via **TRACE CMD CALL *trapname***



CP Tools – TRACE

- **TRACE CLEAR** deletes all traps from current or specified trace set
- **TRACE END** clears all sets
- ***Too*** powerful in many cases:
 - ✓ Use **CLEAR** instead to avoid lost work



CP Tools – TRACE

- Can **TRACE** many instructions which cause SIE exit with (essentially) no overhead
- **MC** in source plus **TRACE MC** makes it easy to find relocated code
- **TRACE DIAG 8** is quick way to find CP commands issued by EXEC or MODULE
- Remember that **TRACE** changes execution at hardware level, so timing-related problems may change/vanish while tracing



CP Tools – TRACE

- **NOSIM** option tells **TRACE** to skip actual execution of some opcodes
 - ✓ **TRACE DIAG 4C RUN NOTERM NOSIM** avoids cutting accounting cards, for example
 - ✓ Beware of programs which complain when they detect “failed” operation due to **NOSIM**
 - ✓ May need to use **CMD** to fake results
- Use **CP SET PFxx NODISP BEGIN** when tracing to avoid typing **b** repeatedly
- Many other capabilities, options



CP Tools – DISPLAY

- **DISPLAY** command displays storage
 - ✓ Options control display contents and format
- **DISPLAY I** translates opcodes, makes code semi-readable
 - ✓ Use to look for “interesting” opcodes when attempting to match code to listing
 - ✓ Look at branches to determine base registers
- **DISPLAY PSW ALL** shows fixed PSW locations (PGMNP SW, etc.)



CP Tools – DISPLAY

- **BASE x , INDEX x** options use GPR x as base/index registers
 - ✓ **D T0.FF;BASEC** shows module header (in standard OS linkage code)
 - ✓ Very powerful as target of **TRACE CMD**
 - ✓ Beware **BASE0/INDEX0** – unintuitively use zero as value, instead of GPR0
- **D T2A0.30** shows last two commands, EXECs, transient modules



CP Tools – DISPLAY

■ **DISPLAY** supports data spaces

- ✓ Operands can indicate primary space, or select by space name, ASIT, etc.
- ✓ EXEC enables **DISPLAY/STORE** from other users via DMSSPCI, DMSPCC, DMSSPCP CSL calls
- ✓ Owner needs **XCONFIG** with **SHARE** in directory
- ✓ Complex **DISPLAY** syntax:

```
D ASIT01C14A0000000001.T2A0.30
```

```
D SPACEPHILS:BASE.T2A0.30
```



CP Tools – LOCATEVM and TRSOURCE

■ CP LOCATEVM

- ✓ Searches user storage for string
- ✓ Can specify in hexadecimal or character
- ✓ Somewhat slow (deliberately, to avoid massive paging), but invaluable at times

■ CP TRSOURCE

- ✓ Performs CP-level tracing
- ✓ Particularly useful for I/O, interrupt problems
- ✓ Discussed in *Fun with CP Debugging Tools*



CP Tools – VMDUMPS

- Poorly understood, often underutilized
- ***Always*** use as follows:
 - ✓ **VMDUMP 0-END DCSS TO ***
 - ✓ Issue from **CP READ**, or via **#CP**
- ***Never*** issue from **RUNNING** or **VM READ** without **#CP**
 - ✓ Dump will be suspect, more difficult to analyze at best; often completely useless



CP Tools – More on VMDUMPS

- Common complaint from user, after service machine ABENDs: “I can’t find the dump”
 - ✓ Use CP **QUERY RDR * ALL XFER ALL**
 - ✓ Shows all reader files created by this userid but owned by another userid
 - ✓ Dump will probably show up owned by **OPERSYMP** or **OPERATNS**



And Don't Forget...

- Many programs have (secret?) built-in options to generate traces, other diagnostics
- Service virtual machines typically perform some sort of tracing and/or logging
 - ✓ Check and save SVM logs with dumps, consoles, etc.
- In rare cases, second-level testing may be appropriate to isolate tracing effects

Problem Scenarios



Basic Scenario: Repeatable, Portable ABEND

- Report: “When I do this, it ABENDs”
- Spool the console, reproduce ABEND
- OK, it fails – now what?
 - ✓ **CP TRACE PROG**, reproduce ABEND
 - ✓ When it stops, look at registers, etc.
 - ✓ Remember to **D T0.2F;BASEC** to view module header (most of the time)
 - ✓ Occasionally, problem is obvious; if not ...



Repeatable, Portable ABEND (continued)

- Look for working variations (options, etc.)
- Simplify test case if possible (remove code)
- Normalize the environment:
 - ✓ ReIPL CMS and rerun
 - ✓ ReIPL again and repeat; presumably load/error addresses, etc. are constant
 - ✓ Create EXEC if necessary to simplify




Repeatable, Portable ABEND (continued)

- **TRACE** code range near ABEND, look for “good” execution of same code
 - ✓ If found, see what’s different
- If execution path diverges near ABEND, try manual fixup (**STORE** and/or **BEGIN**), see if any change
 - ✓ May not work, but other errors generated may be more meaningful



Repeatable, Portable ABEND (continued)

- Take dump *at point of ABEND*; also let product dump if it does so automatically
- Save *both* dumps, plus matching console, offer all to vendor/author
- If you have source, trace back through caller, etc.



Scenario: Non-repeatable ABEND

- Spool the console
- Run with **TRACE PROG CMD VMDUMP** to generate dumps automatically
- Collect any and all dumps
- Examine system activity, other factors at time of ABEND
 - ✓ Look for other anomalies
 - ✓ Try to find pattern, if intermittent
 - ✓ If working examples exist, examine differences



Non-repeatable ABEND (continued)

- If no dump, and not repeatable, report but close problem immediately
 - ✓ Such “anecdotal” reports like this allow tracking intermittent problems
 - ✓ Second (third, fourth, ...) report validates original, may provide new clues leading to resolution



Scenario: Loop

- Spool the console
- If no longer looping, attempt to reproduce
- If not reproducible, treat like non-repeatable ABEND with no dump: report and close
- If still happening, or reproducible, collect tracing information while looping ...



Loop (continued)

- Press PA1 to get to **CP READ** (force disconnect if PA1 not enabled)
 - ✓ Take **VMDUMP** (via **SECUSER** if **SVM**)
 - ✓ **TRACE BR RUN NOTERM PRINT** or
TRACE I RUN NOTERM PRINT
BEGIN
 - ✓ Use **#CP QUERY PRT * ALL** to track output volume generated



Loop (continued)

- Stop after 10,000 or so lines generated:
 - ✓ Press PA1 again
 - ✓ Issue **TRACE END**
 - ✓ Take another VMDUMP
- Pass both dumps, console, trace (printer file) to vendor
- Beware of dumps after PA1: PSW may not reflect loop location – check WAIT bit



Scenario: Error Message

- If true error message, use CMS
`SET TRAPMSG` to generate VMDUMP
- Message header gives clue as to issuer
- Scan source/EXECs/MODULEs, find issuer
- Examine code (if available) to find real message reason
- Beware messages issued for multiple reasons, or from multiple places
 - ✓ May need to trap to find out which case is true

Summary





When All Else Fails ...

- Practice good programming hygiene
 - ✓ Examine user code, fix obvious problems even if not clearly related to problem at hand
 - ✓ Examples: missing **address command** in REXX programs; abbreviations/synonyms used in EXECs
- Release unneeded disks/directories
 - ✓ Old/duplicate program/utility/subfunction may be hiding there



When All Else Fails ...

- Check, re-check, and re-re-check product and user configuration files
- Remember CP directory options which can have far-reaching (and unintuitive) effects!



Conclusion

- CMS and CP are both rich in tools for CMS application debugging
- No one tool is all-powerful (although sometimes one will suffice)
- Learn to use the tools – it makes your job (and mine) easier!



Contact Info

Phil Smith III
Computer Associates International, Inc.
1800 Alexander Bell Drive
Reston, VA 20191

(703) 264-8514 (voice)
(703) 264-8190 (FAX)

Phil.Smith@Sterling.com
USSCIPHS at IBMMail



Resources

- VM Workshop Tapes online
<http://ukcc.uky.edu/~tools>
- VM Workshop home page:
<http://vm.marist.edu/~workshop/>
- VMESA-L (VM discussion list)
Send mail to:
listserv@uafsysb.uark.edu
With body text:
`subscribe vmesa-l`