Tips and Techniques for Debugging Shareable Images

Spring 1992 DECUS Symposium
VA240

Tom Allebrandi (Internet: Allebrandi@Inland.Com)
Chair, VMSnet Working Group, VAX SIG
Senior Engineer, Computer Applications
Research Laboratories, Inland Steel Flat Products Company

A division of Inland Steel Industries

Why are we here?

- Consider the following:
  - A vendor supplies an application that allows user extensions via LIB$find_image_symbol()
  - The vendors application was linked /NODEBUG, /NOTRACEBACK
  - You develop some extension routines and would like to debug them under the vendors application
- How do you do this?

VAX/VMS Shareable Images

- Two types of shareable images
  - Images which are linked with a program
  - Images which are invoked at runtime
- LIB$find_image_symbol() is used to invoke images at runtime
  - This routine will search an image file for a specified symbol
  - If the symbol is found, the image is mapped into memory and the resulting memory address of the symbol is returned

VAX/VMS Shareable Images (continued)

- Many packages provide user extension facilities via LIB$find_image_symbol()
  - "Callable editors"
  - VAX Mail Foreign Transports
  - At least one database vendor

Overview of Image Activation

- Programs are activated in a number of ways
  - Process creation may be given a program image to execute in the new process
  - DCL may cause a program to be activated
    - "RUN" command
    - CLI defined command
    - Foreign command symbols

Overview of Image Activation (continued)

- The Image Activator is a system service that causes an image file to be mapped into memory
  - Phase 1 maps the image into memory
  - Phase 2 adjusts any memory addresses which cannot be determined until the image is mapped

Overview of Image Activation (continued)

- Additional images may be activated automatically
  - When an image is LINKed, it may refer to symbols in other (shareable) images
    - Common or language specific runtime support
    - DECwindows
    - Utility routines such as SORT, CLI, etc

Use of the Image Transfer Vector

- Every image contains four longwords known as the Image Transfer Vector
- The Image Transfer Vector is a list of starting addresses for a program
- The first entry in the list is usually the Image Startup system service
    - This entry is not present if the image was linked /NOTRACEBACK

Use of the Image Transfer Vector (continued)

- Another entry in the list can be LIB$initialize()
  - Programs may define their own routines to be executed before the main program
  - These routines are called by LIB$initialize()
- The "true" main program address is the last entry in the list
- The list is always terminated by one or more zero (0) entries

Use of the Image Transfer Vector (continued)

- Routines which are executed prior to the main program pass control to the next entry on the list when they complete
  - An access violation would occur if a main program attempted to pass control down the chain

The Traceback Handler

- The Image Startup system service checks to see if the Debugger should be started
- If the Debugger is not to be started, a condition handler is established
  - The condition handler will cause a process image dump if one was requested
  - The condition handler will cause a traceback to be generated

The Traceback Handler (continued)

- If Control-Y is typed, followed by the DCL DEBUG command, the Debugger is invoked

- For security reasons, VMS will not let you install an image that invokes the Image Startup system service with privileges
- As noted before, the Image Startup system service is suppressed if the image is linked /NOTRACEBACK

Enabling Traceback

- An image linked /NOTRACEBACK will not have an entry for the Image Startup system service in its Image Transfer Vector
- Patching the Image Transfer Vector to insert an entry for the Image Startup system service will cause tracebacks to be enabled

Enabling Traceback (continued)

- An image linked /NOTRACEBACK will also contain no entry point data
  - A traceback can occur, but there may not be any symbols displayed.
  - The traceback generator will use any traceback information that is available in other images

Enabling Traceback (continued)

- If the main program image is linked /NOTRACEBACK, but any shareable images it uses were linked /TRACEBACK, some traceback information will be present

Example

```
$ patch/absolute t.exe

PATCH Version 4-00 15-Sep-1984


%PATCH-I-NOLCL, image does not contain local symbols
%PATCH-I-NOGBL, some or all global symbols not accessible
PATCH>!
PATCH>!The word at offset 2 contains the offset of the
PATCH>!Image Transfer Vector
PATCH>!
PATCH>examine/word 2
00000002: 0030
PATCH>!
PATCH>!Examine four longwords at that location
PATCH>!
PATCH>examine/long 30:3C
00000030: 00000400
00000034: 00000000
00000038: 00000000
0000003C: 00000000
PATCH>!
PATCH>!Move the first entry, the program start address, to the second entry
PATCH>!
PATCH>deposit/long 34=400
old: 00000034: 00000000
new: 00000034: 00000400
PATCH>!
PATCH>!Set the address of the Image Startup system service as the first entry
PATCH>!
PATCH>deposit/long 30=7FFEDF68
old: 00000030: 00000400
new: 00000030: 7FFEDF68
```

Example (continued)

```
PATCH>!
PATCH>!Take a peek at the new vector
PATCH>!
PATCH>examine/long 30:3C
00000030: 7FFEDF68
00000034: 00000400
00000038: 00000000
0000003C: 00000000
PATCH>!
PATCH>!Make the changes permanent
PATCH>!
PATCH>update
%PATCH-I-WRTFIL, updating image file TA2_ROOT:[TEST]T.EXE;4
PATCH>exit
$
```

The VAX Symbolic Debugger

- Linking a program /DEBUG sets the image header flag IHD$V_LNKDEBUG
  - When the flag is set, the Image Startup system service maps the Debugger into memory and passes control to it
- The Debugger performs whatever initialization it needs to do, then prompts the user for a command
- The Debugger uses the next entry in the Image Transfer Vector to determine the first address to be executed

Enabling the Debugger

- An image linked /NODEBUG will not have the IHD$V_LNKDEBUG flag set
- Patching the image header flags to set the IHD$V_LNKDEBUG bit will cause the Debugger to be invoked

Enabling the Debugger (continued)

- An image linked /NODEBUG will not contain any debug information
  - Debugging can be performed, but there may not be much symbolic information
  - The Debugger will use any debug information that is available in other images
    - If the main program image is linked /NODEBUG, but any shareable images it uses were linked /DEBUG, some debug information will be present

Enabling the Debugger (continued)

- Note the Debugger is not invoked if the main program image was linked /NOTRACEBACK, regardless of the setting of the IHD$V_LNKDEBUG flag
  - Such images must be patched to include the Image Startup system service

Example

```
$ patch/absolute t.exe

PATCH Version 4-00 15-Sep-1984


%PATCH-I-NOLCL, image does not contain local symbols
%PATCH-I-NOGBL, some or all global symbols not accessible
PATCH>!
PATCH>!The image header flags are the longword at offset 20 (hex)
PATCH>!
PATCH>examine/long 20
00000020: 01000028
PATCH>!
PATCH>!The IHD$V_LNKDEBUG flag is the "1" bit
PATCH>!
PATCH>deposit/long 20=01000029
old: 00000020: 01000028
new: 00000020: 01000029
PATCH>!
PATCH>!Take a peek at the new flags
PATCH>!
PATCH>examine/long 20
00000020: 01000029
PATCH>!
PATCH>!Make the change permanent
PATCH>!
PATCH>update
%PATCH-I-WRTFIL, updating image file TA2_ROOT:[TEST]T.EXE;5
PATCH>exit
$
```

Midterm review

- Reconsider the following:
  - A vendor supplies an application that allows user extensions via LIB$find_image_symbol()
  - The vendors application was linked /NODEBUG, /NOTRACEBACK
  - You develop some extension routines and would like to debug them under the vendors application
- How do you do this?

Midterm review (continued)

- Compile and link your routines /DEBUG
- Patch the application to enable traceback
- Patch the application to enable the Debugger
- Run the application
  - The Debugger will use the symbolic information from your sharable image file

Midterm review (continued)

- But but but...
  - "I wanted to examine some variables but they are not there!"
  - "I can’t seem to set any breakpoints!"
- Remember that your image is activated sometime during the execution of the application
  - Neither your program nor the debugging information is available until after the LIB$find_image_symbol() is executed

Invoking the Debugger

- There are a number of ways to invoke the Debugger
  - DCL "DEBUG" command
  - VAX "BPT" instruction
  - Signal SS$_DEBUG

Invoking the Debugger (continued)

- "DEBUG" command
  - If an image has traceback enabled, then
    - Type Control-Y
    - Type "DEBUG"
- This method requires that you know when a good time is to interrupt the program

Invoking the Debugger (continued)

- VAX "BPT" instruction
  - If an image has traceback and debug enabled, then
    - Insert a VAX "BPT" instruction at the point in your code that you would like to have the program stop
    - Run the application
    - Tell Debug to "SET BREAK/EXCEPTION"
    - Start the application

Invoking the Debugger (continued)

- This method requires you to insert some VAX Macro instructions in your code

Invoking the Debugger (continued)

- Signal SS$_DEBUG
  - If an image has traceback and debug enabled, then
    - Insert a LIB$signal(SS$_DEBUG) routine call at the point in your code that you would like to have the program stop
    - Run the application
    - Tell Debug to "SET BREAK/EXCEPTION"
    - Start the application

Invoking the Debugger (continued)

- SS$_DEBUG has a value of 1132 (decimal)

Invoking the Debugger (continued)

- In all of these cases, if your shareable image has been properly compiled and linked with /DEBUG you will be placed in the Debugger with full access to your code and symbols

Installing patched images w/privileges

- Sometimes, the vendors application must be installed with privileges
  - In order to debug user extension code in the proper context, it would be nice to run it under the installed version
  - The Install utility will not allow a progam linked /TRACEBACK to be installed with privileges
    - Install will also disallow the installation of image that has been patched to enable traceback

Installing patched images w/privileges (continued)

- What to do?
- THE FOLLOWING INFORMATION IS TO BE USED AT YOUR OWN RISK
- Actually, Install simply checks to make sure that the address of the Image Startup system service is not in the Image Transfer Vector
  - What if an image could be patched such that the Image Startup system service were invoked, but did not appear in the Image Transfer Vector?

Example

```
$!
$! First we add a routine to the existing image
$!
$ patch t.exe

PATCH Version 4-00 15-Sep-1984

%PATCH-I-NOLCL, image does not contain local symbols
%PATCH-I-NOGBL, some or all global symbols not accessible
PATCH>!
PATCH>!Create some patch space
PATCH>!
PATCH>align /long patch
old patch area size: 00000000
old patch area address: 00000000
new patch area size: 00000200
new patch area address: 00000600
symbol "PATCH" defined as 00000600
PATCH>!
PATCH>!At the patch area, create a small procedure to transfer control
PATCH>!to the Image Startup system service. Note that we don’t know
PATCH>!which registers the service needs so we will save them all.
PATCH>!
PATCH>deposit/word patch=0ff
old: PATCH: 0000
new: PATCH: 00FF
```

Example (continued)

```
PATCH>deposit/instruction patch+2="jmp 7ffedf68+2"
old: 00000602: HALT
old: 00000603: HALT
old: 00000604: HALT
old: 00000605: HALT
old: 00000606: HALT
old: 00000607: HALT
new: 00000602: JMP L^7FFEDF6A
PATCH>!
PATCH>!Peek at our handiwork
PATCH>!
PATCH>examine/word patch
PATCH: 00FF
PATCH>examine/instruction patch+2
00000602: JMP L^7FFEDF6A
PATCH>!
PATCH>!Make the change permanent
PATCH>!
PATCH>update
%PATCH-I-WRTFIL, updating image file TA2_ROOT:[TEST]T.EXE;15
PATCH>!
PATCH>!We need to know the memory address of the patch area
PATCH>!
PATCH>evaluate patch
00000600
PATCH>exit
```

Example (continued)

```
$!
$! Now we need to patch the Image Transfer Vector to insert our little
$! procedure prior to the start of the main program
$!
$ patch/absolute t.exe

 PATCH Version 4-00 15-Sep-1984

%PATCH-I-NOGBL, some or all global symbols not accessible
%PATCH-I-NOLCL, image does not contain local symbols
PATCH>!
PATCH>!The word at offset 2 contains the offset of the
PATCH>!Image Transfer Vector
PATCH>!
PATCH>examine/word 2
00000002: 0030
```

Example (continued)

```
PATCH>!
PATCH>!Examine four longwords at that location
PATCH>!
PATCH>examine/long 30:3C
00000030: 00000400
00000034: 00000000
00000038: 00000000
0000003C: 00000000
PATCH>!
PATCH>!Move the first entry, the program start address, to the second entry
PATCH>!
PATCH>deposit/long 34=400
old: 00000034: 00000000
new: 00000034: 00000400
PATCH>!
PATCH>!Set the address of our routine, which we determined earlier, as the
PATCH>!first entry
PATCH>!
PATCH>deposit/long 30=600
old: 00000030: 00000400
new: 00000030: 00000600
PATCH>!
PATCH>!Take a peek at the new vector
PATCH>!
PATCH>examine/long 30:3C
00000030: 00000600
00000034: 00000400
00000038: 00000000
0000003C: 00000000
```

Example (continued)

```
PATCH>!
PATCH>!Make the changes permanent
PATCH>!
PATCH>update
%PATCH-I-WRTFIL, updating image file TA2_ROOT:[TEST]T.EXE;16
PATCH>exit
$
```

Sample programs

- C source code for the following sample programs is available in the handouts
  - SETTRACE - Enable traceback on an image that does not have it enabled
  - SETDEBUG - Enable debug on an image that does not have it enabled
  - CLEARTRACE - Disable traceback on an image that has it enabled
  - CLEARDEBUG - Disable debug on an image that has it enabled

Sample programs (continued)

- This code will also be available on the Spring 1992 VAX SIG Tape