OpenVMS Technical Journal

V13, July 2009

Table of Contents

Configuring TCP/IP Services for OpenVMS

Bart Zorn

Overview

A technique will be presented to configure TCP/IP Services for OpenVMS for multiple systems in a consistent way, without having to go through all of TCPIP$CONFIG.COM for every system. With this method, changes in the hardware configuration are also easy to handle. No changes are made to the standard TCP/IP software, only two DCL command procedures are added. These procedures do not use any undocumented feature of TCP/IP Services.

Introduction

TCP/IP Services for OpenVMS is not very flexible with regards to changes in the hardware configuration. The TCP/IP interfaces such as we0, ie0, and ie1 are based directly on the corresponding physical devices such as EWA0, EIA0, and EIB0. There is no way to change that relationship using logical names in a similar manner as we are accustomed to doing for most other devices in OpenVMS.

In a real life example, I had to add one Gigabit network adapter to each of four ES47 systems. These ES47 (model 4) systems consist of two 2P boxes and an I/O drawer. Logically, this I/O drawer appears to sit in between the two 2P boxes. (I am not a hardware expert, so I do not know if there are ways to change that.) The new network cards had to be placed in the I/O drawers, because there was no room in either of the two 2P boxes. The result was that the new adapters received a device name somewhere in between the existing ones, and some of the existing ones got a new name! The result of these changes was that I would have to reconfigure TCP/IP services quite extensively. I had anticipated that.

Implementation

Because TCP/IP Services do not allow the use of logical names to designate physical interfaces, a method had to be found to circumvent that.

Two DCL command procedures have been developed and both these procedures contain relevant information for all systems. Identical copies of the procedures can be used on all systems.

The first command should be executed before TCPIP$STARTUP.COM runs. It does the following:

- Sets up logical names to identify all LAN adapters (and their TCPIP alias names) by their hardware MAC address. This is done using the output of the "LANCP SHOW DEVICE/CHAR" command. Of course, this is the infamous technique of parsing the output of a utility, which is prone to cause problems when a new version of that utility provides a different layout. This has actually happened with

the LANCP utility recently! On the other hand, TCP/IP services itself rely on parsing the output of both the TCPIP utility and the LANCP utility.

- Clears out the permanent TCPIP interface database and repopulates it with a default interface described below.
- Clears out any permanent default router information and supplies a new default route described below.

The logical names have the following format:

```
"ADAPTER_00-0F-20-2B-A1-38" = "EWA0","WE0"
```

The default interface and default route for each system are defined in DCL symbols like the following:

```
$ base_interface_<nodename> = -
     "00-0F-20-2B-A1-38 10.5.50.11/24 10.5.50.3"
```

representing the MAC address, the IP address in CIDR format and the default route for this address. This information, combined with the corresponding logical name, is then used to assemble the "TCPIP SET CONFIGURATION INTERFACE" command for this interface. At least one interface needs to be defined because otherwise TCP/IP Services will not start.

Once this DCL procedure has been run, TCP/IP Services can be started and it will operate on one interface.

The second DCL procedure is called by TCPIP$SYSTARTUP.COM. This procedure does two things:

- Configures all interfaces and alias addresses
- Sets and resets the default route

The site where this configuration was developed makes extensive use of alias addresses. It appears that the ifconfig utility is much more flexible and powerful than the "TCPIP SET INTERFACE" command. Ifconfig does not create pseudo interfaces for alias addresses. The drawback is that the "TCPIP SHOW INTERFACE" command cannot display information about aliases which were created with ifconfig.

Another quirk is that ifconfig is supposed to create interface entities. I could not get it working. On the other hand, "TCPIP SET INTERFACE <ifname>" without further information creates the interface if it does not already exist. Conveniently, in that case, it does not issue an error message.

To sum it up, for every interface, a "TCPIP SET INTERFACE" command is issued, to make sure that it exists. All further configurations are done with ifconfig.

Next, for every interface, first the DCL symbol MAC is defined and then for each IP address, a call to a set_interface routine is made:

```
$ MAC := 00-0F-20-2B-A1-38
$ call set_interface 10.5.50.11/24
$ call set_interface 192.168.35.1/24 alias
```

The set_interface routine will figure out which interface is to be defined. This one gets IP address 10.5.50.11, and 192.168.35.1 as alias address.

Of course, DCL symbols can be used instead of constants. At the beginning of this procedure, symbols are defined for all IP addresses that are being used. Several IP addresses are being used more than once, for IP failover or cluster alias purposes.

A side effect of setting an address for an interface is that the default route may be erased. Therefore, once all the interfaces have been defined, the default route is set again. This default route is not necessarily the same as the one defined in the first DCL procedure described above.

The DCL command procedures

The first one is called TCPIP_INIT_CONFIG.COM. It must be called before TCPIP$STARTUP.COM is invoked. I added this procedure to the CONFIG phase of SYSMAN STARTUP, but it can be done in other ways.

```
$ set noon
$ nodename = f$getsyi("nodename")
$ if f$trnlnm("sys$pipe") .nes. "" then goto 'p1'
$!
$ call say_msg "-I- Executing CLUSTER_COMMON:TCPIP_INIT_CONFIG.COM"
$ nodename = f$getsyi("nodename")
$ debug = p1 .nes. ""
$ icalc := $sys$tools:icalc ! freeware tool, used for mask calculation
$ saved_parse_style = f$getjpi("","parse_style_perm")
$ set process/parse=traditional ! Needed for icalc, a ^ is being used
$ this_proc = f$environment("procedure")
$!
$! TCPIP_INIT_CONFIG.COM
$!
$! 31-Aug-2006, Bart Zorn
$!
$! This procedure is called before TCPIP$STARTUP.COM and does three things:
$!
$! 1. It sets up logical names to identify all lan adapters (and their TCPIP
$!    alias names) by their hardware MAC address. In addition, logical names
$!    that are common to all systems are defined.
$!
$! 2. It clears out the permanent tcpip interface database and repopulates
$!    it with the default interface defined below.
$!
$! 3. It clears out any permanent default router information and supplies
$!    a new default route defined below.
$!
```

```
$ base_interface_node01 := 00-0F-20-2B-A1-38 10.5.50.11/24   10.5.50.3
$ base_interface_node02 := 00-0B-CD-F4-E4-A8 10.5.50.12/24   10.5.50.3
$!
$! 1. Lookup all hardware MAC addresses
$!
$ pipe mcr lancp show device/characteristics | -
        search sys$pipe "Device Characteristics","Hardware LAN address" | -
            @ 'this_proc' do_define
$!
$! 2a. Delete current permanent interface configuration
$!
$ pipe tcpip show configuration interface/full -
      > sys$manager:tcpip_saved_configuration.txt 2> nl:
$ pipe tcpip set configuration nointerface */noconfirm > nl: 2> nl:
$!
$! 2b. Repopulate the configuration database
$!
$ lognam = "adapter_" + f$element(0," ",base_interface_'nodename')
$ interface = f$trnlnm(lognam,,1)
$ if interface .eqs. ""
$   then
$     call say_msg -
      "-F- TCP/IP default interface is not defined. TCP/IP will not startup."
$     goto exit
$   endif
$ ip_address = f$element(1," ",base_interface_'nodename')
$ mask_length = f$element(1,"/",ip_address)
$ ip_address = f$element(0,"/",ip_address)
$ call generate_mask mask_length mask_longword
$ call convert_longword_to_address mask_longword mask_address
$ vf = f$verify(1)
$ tcpip set configuration interface 'interface' -
      /host='ip_address' /network_mask='mask_address'
$ ! 'f$verify(vf)'
$!
$! 3a. Delete current permanent routing configuration, ignoring errors.
$!
$ pipe tcpip show route/permanent > sys$manager:tcpip_saved_routing.txt 2> nl:
$ pipe tcpip set noroute/permanent/noconfirm/gate=* > nl: 2> nl:
$!
$! 3b. Define permanent default route information
$!
$ def_route = f$element(2," ",f$edit(base_interface_'nodename',"compress,trim"))
$ vf = f$verify(1)
$ tcpip set route/gateway='def_route'/default/permanent
$ ! 'f$verify(vf)'
$!
$exit:
$ set process/parse='saved_parse_style'
$ exit
$!
$say_msg: subroutine
$ msg = f$fao("!8%T !AS",0,P1)
$ write sys$output msg
$ if f$trnlnm("sys$output") .nes. f$trnlnm("sys$error") then write sys$error msg
$ endsubroutine
$!
$do_define:
$!
$! Read the first line, it contains the device name
```

```
$!
$ read/end=eof sys$pipe line
$ device = f$element(2," ",line)
$!
$! The second line contains the physical address
$!
$ read/end=eof sys$pipe line
$ address = f$element(0," ",f$edit(line,"trim,compress"))
$!
$! Build the TCP/IP style interface name from the device name
$! This assumes that we have no more than 6 devices of any given type
$!
$ interface = f$extract(1,1,device) + f$extract(0,1,device)
$ unit = %X'f$extract(2,1,device)' - 10
$ interface := 'interface''unit'
$!
$ vf = f$verify(1)
$ define/system/exec/nolog adapter_'address' 'device','interface'
$ ! 'f$verify(vf)'
$ goto do_define
$eof:
$ exit
$!
$generate_mask: subroutine
$ i = 32 - 'p1'
$ pipe icalc 2^'i' > nl:
$ 'p2' == .not. ('ICALC_OUT' - 1)
$ endsubroutine
$!
$convert_longword_to_address: subroutine
$ hex_longword = f$fao("!XL",'p1')
$ a1 = %x'f$extract(0,2,hex_longword)'
$ a2 = %x'f$extract(2,2,hex_longword)'
$ a3 = %x'f$extract(4,2,hex_longword)'
$ a4 = %x'f$extract(6,2,hex_longword)'
$ 'p2' :== 'a1'.'a2'.'a3'.'a4'
$ endsubroutine
```

The second procedure is called CREATE_INTERFACES.COM. It is called from
TCPIP$SYSTARTUP.COM.

```
$!
$! CREATE_INTERFACES.COM
$!
$! 24-Mar-2003, Bart Zorn
$!
$ SET NOON
$ CALL SAY_MSG "-I- Executing CLUSTER_COMMON:CREATE_INTERFACES.COM"
$ ifconfig := $tcpip$ifconfig
$ nodename = f$getsyi("nodename")
$!
$! Define symbols to be used here
$!
$ NODE01 := 10.5.50.11/24
$!
$ GOTO SYSTEM_'NODENAME'
$ EXIT              ! Do not fall through
$!
```

```
$SYSTEM_NODE01:
$!
$ MAC := 00-0F-20-2B-A1-38
$ call set_interface 'NODE01'
$ call set_interface 192.168.35.1/24 alias
$!
$ MAC := 00-0F-20-2B-A1-37
$ call set_interface 'NODE01'
$ call set_interface 192.168.35.1/24 alias
$!
$ goto exit
$!
$SYSTEM_NODE02:
$!
$ MAC := 00-0B-CD-F4-E4-A8
$ call set_interface 10.5.50.12/24
$ call set_interface 10.5.50.18/24 alias          ! Cluster alias
$!
$ MAC := 00-0B-CD-F4-E4-A9
$ call set_interface 10.5.50.12/24
$ call set_interface 10.5.50.18/24 alias          ! Cluster alias
$!
$ MAC := 00-08-02-91-88-CA
$ call set_interface 10.5.50.13/24
$ call set_interface 10.5.50.33/24 alias
$!
$ call set_interface 10.5.52.1/24 home            ! Apllication alias
$!
$ goto exit
$!
$EXIT:
$ gosub reset_default_route
$ EXIT
$!
$say_msg: subroutine
$ msg = f$fao("!8%T !AS",0,p1)
$ write sys$output msg
$ if f$trnlnm("sys$output") .nes. f$trnlnm("sys$error") then -
      write sys$error msg
$ endsubroutine
$!
$set_interface: subroutine
$!
$! p1 - address/mask
$! p2 - optional parameters
$! p3 - additional parameters for ifconfig
$!
$ lnm = "ADAPTER_" + MAC
$ interface = f$trnlnm(lnm,,1)
$ if interface .eqs. ""
$   then
$     call say_msg "-E- ''lnm' logical name is missing"
$     exit
$   endif
$!
$! Create interface if it does not already exist
$!
$ tcpip set interface 'interface'
$!
$ if f$edit(p2,"lowercase") .eqs. "home" then p2 := home alias
```

```
$ params = f$edit(f$fao("!AS !AS !AS",p3,interface,p2),"trim,compress,lowercase")
$ sv = f$verify(1)
$ ifconfig 'params' 'p1'
$ ! 'f$verify(sv)'
$ endsubroutine
$!
$reset_default_route:
$ if "''new_default_route'" .eqs. "" then return
$ if f$mode() .eqs. "INTERACTIVE"
$   then
$      if f$trnlnm("tt") .nes. "OPA0:" then return
$   else
$      if f$mode() .nes. "OTHER" then then return
$   endif
$!
$ pipe tcpip netstat -rn | search sys$pipe default | -
      ( read sys$pipe line ; define/job/nolog line &line )
$ line = f$edit(f$trnlnm("line"),"trim,compress")
$ deassign/job line
$ default_present = f$element(0," ",line) .eqs. "default"
$ if default_present
$   then
$      current_default_route = f$element(1," ",line)
$      if current_default_route .eqs. new_default_route then return
$   endif
$ vf = f$verify(1)
$ tcpip set route /gate='new_default_route' /default
$ ! 'f$verify(vf)'
$ if default_present
$   then
$      if current_default_route .nes. new_default_route
$        then
$           vf = f$verify(1)
$ tcpip set noroute /gate='current_default_route' /noconfirm
$           ! 'f$verify(vf)'
$        endif
$   endif
$ return
```

Things yet to be done

When a new system needs to be configured, it is still necessary to run
TCPIP$CONFIG.COM once. I have not yet reverse engineered what steps
TCPIP$CONFIG.COM takes with regard to the host name and domain name settings.
Also, the client and server configuration needs to be done with TCPIP$CONFIG.COM.

Summary

The techniques described here allow for a complete interface configuration for TCP/IP
Services for OpenVMS with two DCL command procedures. These procedures are
organized in such a way that they contain all relevant information for all systems to be
configured. This makes it a lot easier to configure many systems and prevent duplicate IP
addresses and other errors.

The author can be contacted at Bart.Zorn@Yahoo.com

OpenVMS I64 TIE Internals: Emulating Alpha Control Instructions

Tim E. Sneddon

Preface

This article is the first of what will hopefully become a series of articles that describe the internals of the OpenVMS I64 Translated Image Environment. In subsequent issues of the OpenVMS Technical Journal it is expected that there will be further articles covering topics such as:

- stack walking and unwinding;
- floating-point emulation;
- the Alpha instruction emulator;
- TIE initialization and setup; and
- image translation.

These are but a few of the topics that are likely to be covered in future articles. Suggestions are also always welcome. See the end of this document for details on contacting the author.

Conventions

Table 1 lists some of the conventions used in this article.

Convention	Meaning
Foreign	This term is used to describe any code that cannot be run natively on OpenVMS I64.
Native	This term describes any code that runs natively on OpenVMS I64
AEST	Alpha Environment Software Translator. The binary translation utility that generates OpenVMS I64 images from OpenVMS Alpha images.
VEST	VAX Environment Software Translator. The binary translation utility that generates OpenVMS Alpha images from OpenVMS VAX images.
TIE	Translate Image Environment. The name given to the execution environment that allows user-mode programs from different OpenVMS supported architectures to be executed.
Procedure Descriptor	This term refers to an OpenVMS Alpha procedure descriptor. This is described in the Starlet module $PDSCDEF.
Function Descriptor	This term refers to an OpenVMS I64 function descriptor. This is described in the Starlet module $FDSCDEF.

Table 1 Conventions Used in this Article

What Has The Translator Generated?

Using certain qualifiers on the AEST translation utility it is possible to observe what happens to the original OpenVMS Alpha image after it has been translated. Figure 1

demonstrates some of the output that can be seen. This example shows the correlation between the original Alpha instructions and the equivalent Itanium instructions.

```
0002006C   BIS          R31,#255,R17    | 00000000000040230h:   { 0:   120080fe840  M        addl    r33 = 0ffh, r0
00020070   BIS          R31,#10,R16     |                         1:   00008000000  F        nop.f  00h
00020074   MULQ         R16,R17,R17     |                         2:   12000014800  I        addl  r32 = 0ah, r0 ;;
00020078   ZAP          R17,#253,R17    |                         dispersal:   0d
0002007C   BIS          R31,#2,R25      |
00020080   LDA          R16,00B3(R2)    | 00000000000040240h:   { 0:   0c708040c80  M        setf.sig  f50 = r32
00020084   LDQ          R26,00D0(R2)    |                         1:   0c708042cc0  M        setf.sig  f51 = r33
00020088   LDQ          R27,00D8(R2)    |                         2:   00008000000  I        nop.i  00h ;;
0002008C   JSR          R26,(R26)       |                         dispersal:   09
                                        |
                                        | 00000000000040250h:   { 0:   00008000000  M        nop.m    00h
                                        |                         1:   1d19b200d00  F        xma.l  f52 = f50, f51, f0
                                        |                         2:   00008000000  I        nop.i  00h ;;
                                        |                         dispersal:   0d
                                        |
                                        | 00000000000040260h:   { 0:   08708068840  M        getf.sig  r33 = f52
                                        |                         1:   00000000000  L        movl  r92 = 0ff00h ;;
                                        |                         2:   0cff0001700  L
                                        |                         dispersal:   05
                                        |
```

Figure 1 Listing of Alpha Instructions and Corresponding Translated Itanium Instructions

This output was generated using the command:

```
$ AEST/VERBOSE/LIST/DUMP=IA64=ALL <image>
```

For more information on the qualifiers accepted by AEST and how to use them, refer to the online DCL help.

Observing Image Execution

With the help of some debugging features built into the TIE run-time library it is possible to observe the execution of the translated image. By defining the logical PRTCHK_PRT_ALL to "1" TIE$SHARE will begin producing a trace to SYS$OUTPUT. If it is inconvenient to have this output written to SYS$OUTPUT, it is possible to define the logical PRTCHK_FILE to an alternate output file. Not all routines generate output, but the important ones do. It is also possible to enable interpretation of all images, even those that had native code generated as part of the translation process. To do this, define the logical TIE$INTERPRET to "1". Figure 2 shows a portion of the output from a simple MACRO-64 program that executes a collection of ZAP and ZAPNOT instructions. It is annotated to give a brief description of the operations taking place.

```
    IAS control flow, magic : 6200 (1)
FUNC CALL: tie$xxxx_lookup, lookup_addr = 000000000002006C (2)
    Address is in TRANSLATED image : ZAP
FUNC RET: tie$xxxx_lookup, returns address : 000000000002006C, TIE$CODE_AXP, new GP 0000000000000000
FUNC CALL: tie$is_jump_to_native, AXP_R27 = 000000007B93C7D0 target_pc = 000000000002006C
FUNC RET: tie$is_jump_to_native, rc = 0
FUNC CALL: tie$get_AXP_FPCR_initial_value
    ieee_fpcr : 0000000000000000
FUNC RET: tie$get_AXP_FPCR_initial_value, fpcr 0000000000000000
FUNC CALL: tie_alpha_emulate, asb->pc = 000000000002006C (3)
    000000000002006C  47FFF411  BIS              R31,#255,R17
    0000000000020070  47E15410  BIS              R31,#10,R16
    0000000000020074  4E110411  MULQ             R16,R17,R17
    0000000000020078  4A3FB611  ZAP              R17,#253,R17
    000000000002007C  47E05419  BIS              R31,#2,R25
    0000000000020080  220200B3  LDA              R16,00B3(R2)
    0000000000020084  A74200D0  LDQ              R26,00D0(R2)
    0000000000020088  A76200D8  LDQ              R27,00D8(R2)
    000000000002008C  6B5A4000  JSR              R26,(R26) ;hint=0000, likely PC=0000000000020090
     R27 is pointing to 000000007B93C7D0 contents FFFFFFFF849986C0
     R26 is pointing to 000000007B93C7D8 contents 000000007BB1A000
    Bingo we have a match all is well - r27==asb_pc (4)
FUNC RET: tie_alpha_emulate, TIE_K_JUMP/BRANCH , jump to : FFFFFFFF849986C0
    IAS control flow, magic : 8100 (5)
```

Figure 2 TIE Run-Time Library Trace Output

1. Execution has just transferred to the TIE$AXP_JUMP_TO glue routine.
2. A lookup is being performed on the address 000000000002006C to determine what kind of code is to be executed.
3. It was determined the code was native Alpha and no equivalent translated code was found. Therefore the Alpha instruction emulator is called and used to execute the code.
4. The last instruction to execute was a JSR. The emulator is now determining how to proceed with execution.
5. The TIE is now looking up the return address specified in the JSR instruction to determine how to proceed.

Note: AEST also responds to the PRTCHK_PRT_ALL logical and generates quite a bit of output. Before translating an image it is usually a good idea to deassign this logical.

Alpha Register Mapping

To make it easier to understand some of the Itanium code samples, the following table shows the static register mapping used by the TIE to maintain the Alpha register file.

Alpha Register	Usage	Itanium Register
R0	Function value register	R101
R1	Conventional scratch register	R102
R2-R15	Conventional saved registers	R54-R67
R16-R21	Argument registers	R32-R37
R22-R24	Conventional scratch registers	R103-R105
R25	Argument information (AI) register	R106
R26	Return address (RA) register	R68
R27	Procedure value (PV) register	R69
R28	Volatile scratch register	R107

Alpha Register	Usage	Itanium Register
R29	Frame pointer (FP) register	R70
R30	Stack pointer (SP) register	R12
R31	ReadAsZero/Sink (RZ) register	R0[1]
F0-F1	Floating-point function value register	F32-F33
F2-F9	Conventional saved registers	F16-F23
F10-F15	Conventional scratch registers	F34-F39
F16-F21	Argument registers	F8-F13
F22-F30	Conventional scratch registers	F40-F48
F31	ReadAsZero/Sink register	F0[1]
MBPR	Mailbox pointer register	R72
FPCR	Floating-point control register	R73
PS	Processor status register	R74
PC	Program counter	R75
	Internal TIE scratch register	R3, R21-R24, R26-R31
	Internal TIE local registers	R76-R80
	Internal TIE translator flag register	R82
	Internal TIE output registers	R108-R115

Table 2 Alpha Register Mapping

If some of these mappings don't appear to follow a logical order it is because many of register mappings were changed late in the design of TIE. Originally it was intended that Alpha registers sharing similar functions as Itanium registers would be mapped together (as was done with VAX registers on the Alpha platform). However, this was eventually changed and so now all Alpha registers (with the exception of the stack pointer) exist in the register stack frame.

The mapping of VAX registers can be found in *OpenVMS Alpha Internals and Data Structures: Scheduling and Process Control*. The mappings have been retained, so all VAX registers map to their original Alpha registers, which in turn map to the equivalent Itanium register.

Introduction

The Translated Image Environment (TIE) is the support environment which executes user mode images compiled and linked on OpenVMS VAX and OpenVMS Alpha that have been subsequently translated for execution on the OpenVMS I64 platform. The

---

[1] Read-only. Writing to Itanium register R0 or F0 results in an Illegal Operation fault.

translation is achieved using the Alpha Environment Software Translator (AEST) and, in the case of OpenVMS VAX images, the VAX Environment Software Translator (VEST) binary translation tools. While the VEST translator is only available on OpenVMS Alpha, it is supported to translate a VEST'd image using the AEST translator.

These translation utilities generate native images that work with the TIE run-time library to emulate a native OpenVMS Alpha or VAX environment. It is the responsibility of the translator to present the original image in such a fashion that the TIE run-time library can then execute the foreign code. In most cases these utilities are able to generate equivalent native code from the foreign code. It is the branching inside and between these environments that is the focus of this article.

In some areas VAX support is touched on. However, the main focus of this article is the support of the Alpha control instructions. The *OpenVMS Alpha Internals & Data Structures* manual is still a relevant reference as the TIE, present on OpenVMS Alpha for the support of the OpenVMS VAX environment, has been ported to OpenVMS I64 with minimal changes.

Taking a JuMP...

The Alpha architecture presents a collection of closely related control instructions. In the Alpha Architecture Handbook, these are divided as 'Conditional Branch', 'Unconditional Branch' and 'Jumps'. While this division is important to the TIE, a more relevant way to divide them is 'Local' and 'Non-Local'.

Although the two types of branch are handled differently in how they obtain their addresses, all branches within the Alpha environment use the regular conditional branch, 'br', instruction. This is because the emulated environment is contained within a regular OpenVMS I64 frame. The conditional procedure call, or 'br.call', instruction is only used for native calls.

This does make the process of stack walking and unwinding non-trivial. However, by avoiding a true procedure call the emulated Alpha registers continue to be available across Alpha calls. There is no need to consider the consequences of the 'alloc' instruction and how to maintain context. The only time this needs to be considered is when switching environments and that is handled by jacketing procedures, discussed later in this article[2].

Local Branches

Local branches are described as those that are capable of branching forward or backwards a PC relative distance of +/-1M instructions. It also happens that this encompasses all conditional branch instructions. Table 3 summarizes the local control instructions.

---

[2] See the section 'Switching Environments'.

Mnemonic	Operation
BEQ	Branch if Register Equal to Zero
BGE	Branch if Register Greater Than or Equal to Zero
BGT	Branch if Register Greater Than Zero
BLBC	Branch if Register Low Bit is Clear
BLBS	Branch if Register Low Bit is Set
BLE	Branch if Register Less Than or Equal to Zero
BLT	Branch if Register Less Than Zero
BNE	Branch if Register Not Equal to Zero
BR	Unconditional Branch
BSR	Branch to Subroutine

Table 3 Local Alpha Control Instructions

These instructions are usually used within a single object module to handle local branching. As such when it comes to translating these instructions to a native instruction sequence there is almost always no execution time lost in the resulting image. This is due to the fact that the branch target is known at the time of translation. However, if the destination of these branch instructions cannot be located within the image being translated then the instruction will be emulated by TIE run-time library via the Alpha instruction emulator.

For all non-emulated branches a small native instruction sequence is generated. Figure 3 shows the sequence generated by AEST when translating an Alpha BNE instruction, it is annotated below.


```
BNE    R1,000018                         cmp.eq  p33, p32 = r0, r102  ①

                             (p32) adds     r75 = 044h, r75           ②

                             (p33) adds     r75 = 02ch, r75           ③

                             (p32) br.cond.spnt.few  $+0c0h           ④
```

Figure 3 Itanium Code Generated for Alpha BNE Instruction

1. Here the compare part of the BNE instruction is performed.
2. In the event that the comparison yields a positive result, the Alpha PC is set to the address of the branch target.
3. If the comparison is not true, the Alpha PC is updated to point to the next instruction, following the BNE instruction.
4. Lastly, the branch is taken if the result was positive.

All other conditional branches are of the same format, simply substituting the relevant compare instruction relation as necessary.

For the unconditional branches it is even simpler. Figure 4 shows the sequence generated by AEST for a BSR instruction and is annotated below. The only difference between it and a BR instruction is that BSR sets up a return address.

Figure 4 Itanium Code Generated for Alpha BSR Instruction

1. Here the emulated Alpha R26 register is configured with the return emulated PC.
2. The emulated Alpha PC is then updated to point to the first instruction of the destination.
3. Lastly, the branch is taken.

Non-Local Branches

Non-local branches are described as those that take a register argument containing an absolute address, rather than a PC offset. Table 4 summarises the non-local control instructions.
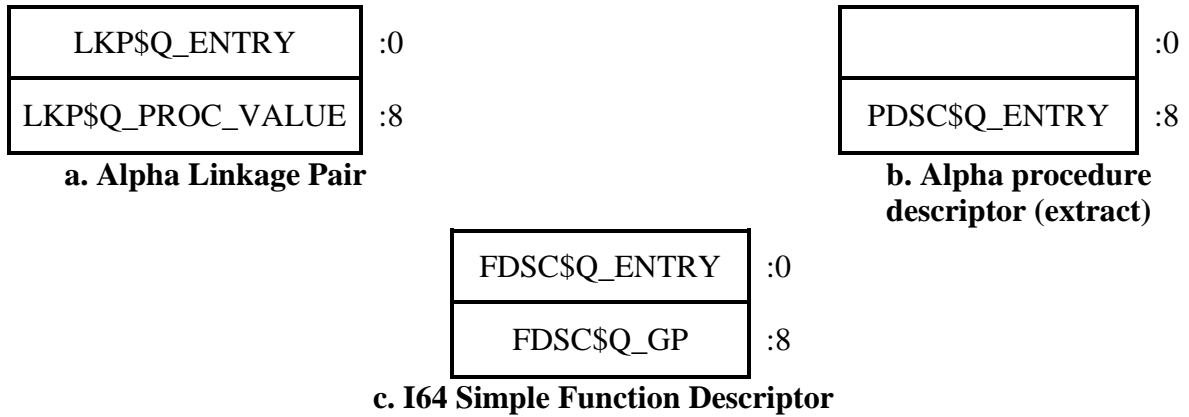
Mnemonic	Operation
JMP	Jump
JSR	Jump to Subroutine
RET	Return from Subroutine
JSR_COROUTINE	Jump to Subroutine Return

Table 4 Non-Local Alpha Control Instructions

The emulation of these instructions does incur quite a bit of overhead as the destination is assumed to be unknown at translation time. This means that these instructions cannot be handled with a small instruction sequence like local branches. It requires support from the TIE run-time library. This comes in the form of the routine TIE$AXP_JUMP_TO. This routine is the first stop on all non-local branches with the exception of JSR. All JSR instructions first branch to TIE$AXP_JSR_TO before falling through to the routine, TIE$AXP_JUMP_TO.

JSR instructions require special pre-processing by TIE$AXP_JSR_TO because in some cases the destination PC for input to the JSR instruction is fetched from a procedure descriptor (see Figure 5b) and not a linkage pair (see Figure 5a). This causes problems when the destination is a native routine as the procedure value register then points to a native function descriptor (see Figure 5c). As can be seen from the illustrations there is no problem in the instance of a function descriptor being mistaken for a linkage pair as the offset to the address of the procedure entry point is the same. However, in the case of the function descriptor being mistaken for a procedure descriptor the global data pointer

(GP) for the native image is then loaded and used as the destination PC. Figure 6 compares the two instruction sequences.

LKP$Q_ENTRY	:0
LKP$Q_PROC_VALUE	:8

a. Alpha Linkage Pair

	:0
PDSC$Q_ENTRY	:8

b. Alpha procedure descriptor (extract)

FDSC$Q_ENTRY	:0
FDSC$Q_GP	:8

c. I64 Simple Function Descriptor

Figure 5 Alpha and I64 Procedure Descriptors

To remedy this when the TIE walks the list of activated images (IAC$GL_IMAGE_LIST) during initialisation gathering details for its own internal list of native images, it also gathers up the GPs and caches them in another list (TIE$CACHED_GPs). When TIE$AXP_JSR_TO is called it attempts to match the destination PC with an entry in this list. If a match is found, then the destination PC is altered by fetching the real entry point address from the function descriptor pointed at by the Alpha register, R27 (procedure value register). At this point TIE$AXP_JSR_TO then falls through to TIE$AXP_JUMP_TO and continues as normal.

```
LDQ    R27, 20(R27)  ; Fetch procedure      LDQ    R26, 20(R4)    ; Fetch procedure entry
descriptor                                  point
LDQ    R26, 08(R27)  ; Fetch procedure entry LDQ   R27, 28(R4)    ; Fetch procedure
point                                       descriptor
JSR    R26, (R26)    ; Call procedure        JSR    R26, (R26)     ; Call procedure
```

Figure 6a. Fetching Entry Point From Descriptor   b. Fetching Entry Point From Linkage Pair

Once the address lookup has been performed, TIE$AXP_JUMP_TO then transfers control appropriate to the type of address being branched to. This may mean starting the Alpha instruction emulator, calling a native routine, branching to another translated Alpha routine or to a pseudo image.

Finding A Place To Go

Jumping to an address may be one thing. However, locating that address to determine how to jump is entirely something else. For the Alpha environment, all call address lookups go through the routine TIE$XXXX_LOOKUP. For the VAX environment the equivalent routine is TIE$VESTED_LOOKUP. It is these routines that have the job of determining what kind of code is at the specified address and locating any translated code that may be associated with it.

For the rest of this section, TIE$XXXX_LOOKUP refers to both the Alpha environment TIE$XXXX_LOOKUP and the VAX environment TIE$VESTED_LOOKUP, unless otherwise specified.

Performing the Lookup

An address lookup begins with TIE$XXXX_LOOKUP first checking that the address being looked up is not the special return address used when returning from native code. In this case the lookup terminates with a return status of TIE$CODE_N2T_RETURN and the caller (usually TIE$ALPHA_TO_IPF or TIE$VAX_TO_IPF) will then begin the process of translating arguments back to their respective environments.

The lookup continues by first checking that this address has not previously been requested. This is done by checking the lookup cache (TIE$GR_LOOKUP_CACHE). To speed up the process of looking up a call address, all successful address lookups and their results are stored in a 4096 entry hash table. This table is managed using the FNV-1a[3] hash algorithm. In the event that no match is found the lookup routine begins walking internal TIE data structures.

The process starts by first looking up the list of pseudo images loaded by the TIE during initialization. A pseudo image, as the name implies, is not a real image. It exists only as a collection of data structures in memory. Its purpose is to allow the native TIE to intercept lookups by translated or emulated code to TIE routines from previous TIE environments, such as the TIE$SHARE image used to support the VAX environment under OpenVMS Alpha. An example of this is the substitution of the OpenVMS Alpha routine OTS$CALL_PROC with the internal TIE service, TIE$$AXP_OTS$CALL_PROC. The original OTS$CALL_PROC from OpenVMS Alpha has no relevance on OpenVMS I64, so the TIE$$AXP_OTS$CALL_PROC service has been written to act as a substitute. It is simply a wrapper that jumps to TIE$AXP_JUMP_TO.

base_va
end_va
fp_array
fp_array_sz
imcb$t_log_image_name
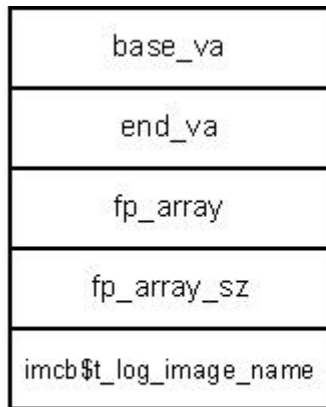
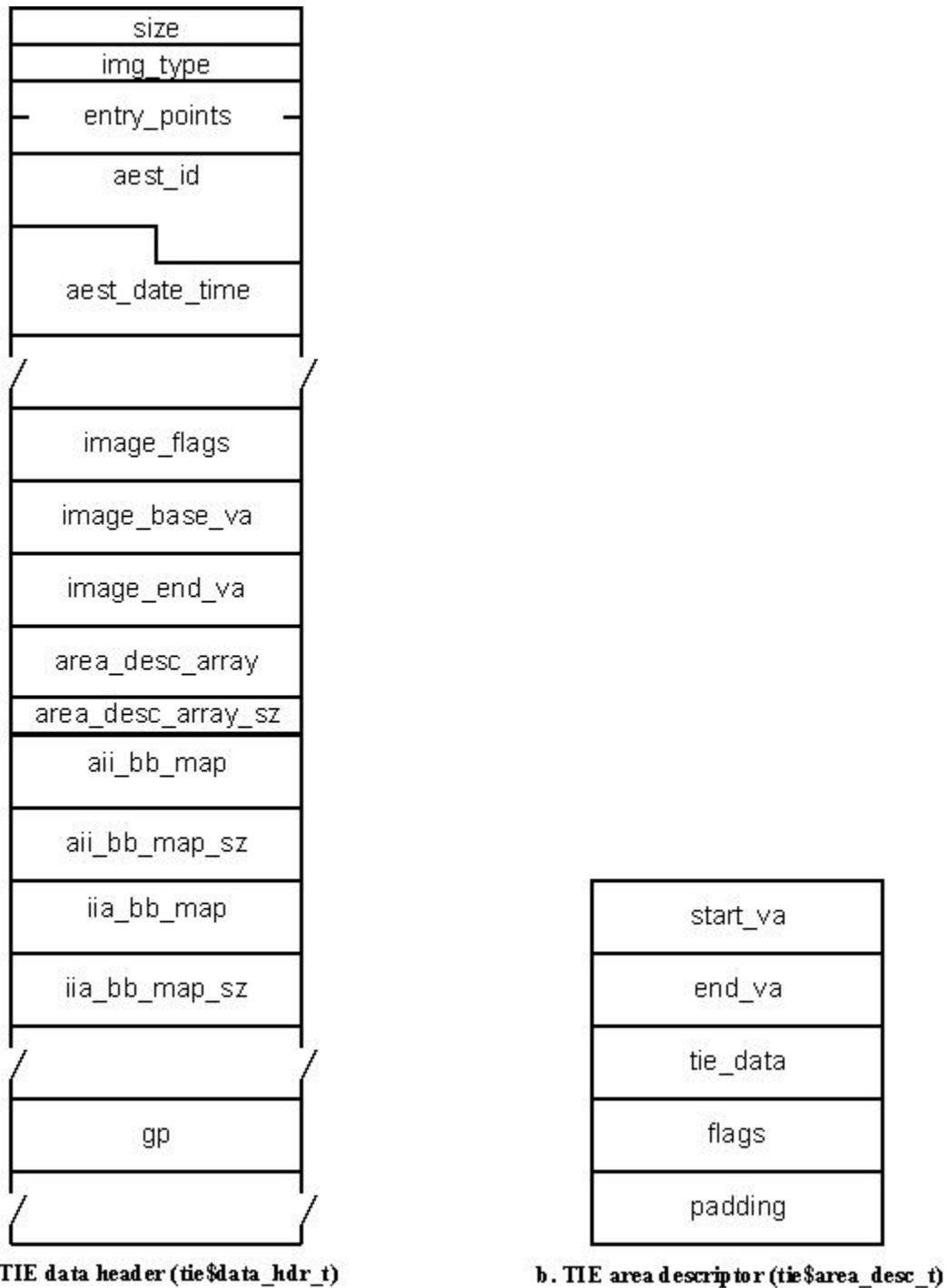| func |
| --- |
| name |
| type |

a. TIE Pseudo Image Header
(tie$pseudo_img_hdr_t)
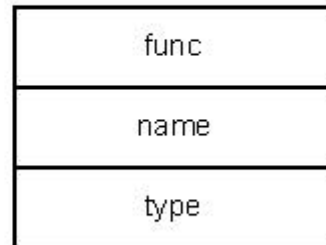
b. TIE Pseudo Function Descriptor
(tie$pseudo_func_t)

Figure 7 TIE Pseudo Image Data Structures

Like almost all internal image data blocks the list of pseudo images is stored in a B-tree. The root of this tree is TIE$PAXP_IMG_DESC_ROOT. Each node in the tree contains a pointer to a TIE pseudo image header (see Figure 7a). These structures are queried using

---

[3] See the section 'For more information' for details of the Fowler Noll Vo hash algorithm.

the routine TIE$FIND_PSEUDO_IMG. In the event that the call address being looked up exists in the address space described by the pseudo image, then the routine TIE$FIND_PSEUDO_FP attempts to locate a matching function.

size
img_type
entry_points
aest_id
aest_date_time
image_flags
image_base_va
image_end_va
area_desc_array
area_desc_array_sz
aii_bb_map
aii_bb_map_sz
iia_bb_map
iia_bb_map_sz
gp

a. TIE data header (tie$data_hdr_t)

start_va
end_va
tie_data
flags
padding

b. TIE area descriptor (tie$area_desc_t)

Figure 8 TIE Internal Image Descriptors

The next step is to try and find the call address in the list of loaded translated images. This too is a B-tree list. The root of this tree Is TIE$IMG_DESC_ROOT and is queried

using the routine TIE$FIND_IMG. If the call address exists in the address space of this image, then TIE$FIND_IMG returns a pointer to the TIE data header (see Figure 8a). TIE$FIND_IMG_AREA then uses this structure to locate, using a binary search, the specific area containing the call address and return the corresponding TIE area header (see Figure 8b). The type of code pointed at by the call address can then be determined based on the flags field of the TIE area header. Table 5 shows possible values for the flags field.

Value	Symbolic Name	Meaning
1	tie$vax_code_area_fl	Section contains VAX instructions
2	tie$axp_code_area_fl	Section contains Alpha instructions
4	tie$ipf_code_area_fl	Section contains Itanium instructions
8	tie$axp_nonshraddr_area_fl	Section contains Alpha non-shareable address data

Table 5 Tie Area Descriptor Type Flags

In the case that the call address is found to exist in an area containing foreign code (the flag tie$axp_code_area_fl is set) TIE$XXXX_LOOKUP calls TIE$FIND_AII_BB. This routine performs a binomial search on the list of basic blocks pointed to by the field aii_bb_map. This field points to an array of octawords containing the mapping between Alpha basic blocks and their corresponding native translated blocks. The first quadword contains the address of the Alpha block and the second contains the address of the corresponding native block. It is also possible to perform a reverse mapping using the array pointed at by the iia_bb_map field.

Return Status	Meaning
TIE$CODE_IPF_TRANSLATED TIE$CODE_VAX_TRANSLATED[#]	The lookup address either points to translated native code, or a foreign basic block that had a corresponding translated block and TIE$INTERPRET is not active. The address returned points to translated native code.
TIE$CODE_VESTED_AXP[#]	The lookup address points to VAX code that has corresponding translated Alpha code, but no valid native translated code. The return address points to the Alpha code.
TIE$CODE_AXP TIE$CODE_VAX	The lookup address points to foreign code that either has no corresponding translated code, or TIE$INTERPRET is in effect. The return address matches the lookup address.
TIE$CODE_OUTSIDE	The lookup address points to code outside of any translated image. The return address matches the lookup address.
TIE$CODE_PSEUDO_AXP TIE$CODE_PSEUDO_VAX[#]	When the lookup address is in a pseudo image.

Return Status	Meaning
TIE$CODE_VAX_SYS_SERV[#]	When the lookup address points to a VAX system service entry point. The returned address is the corresponding native system service entry point.
TIE$CODE_AXP_TIE[#]	The lookup address points to an Alpha TIE image.
TIE$CODE_N2T_RETURN	The lookup address points to the return address used by TIE$NATIVE_TO_TRANSLATED

[#]These status codes are returned only by TIE$VESTED_LOOKUP.

Table 6 TIE$XXXX_LOOKUP and TIE$VESTED_LOOKUP Return Codes

For the VAX environment lookups are performed using the routine TIE$FIND_VAX_BB. This routine performs a binomial search of the basic block mapping list pointed at by viai_bb_map in the TIE data header (not shown in Figure 8a).

In the event that no match is found at all TIE$XXXX_LOOKUP assumes that the address is native and returns the status TIE$CODE_OUTSIDE. Table 6 shows possible values returned by TIE$XXXX_LOOKUP and TIE$VESTED_LOOKUP.
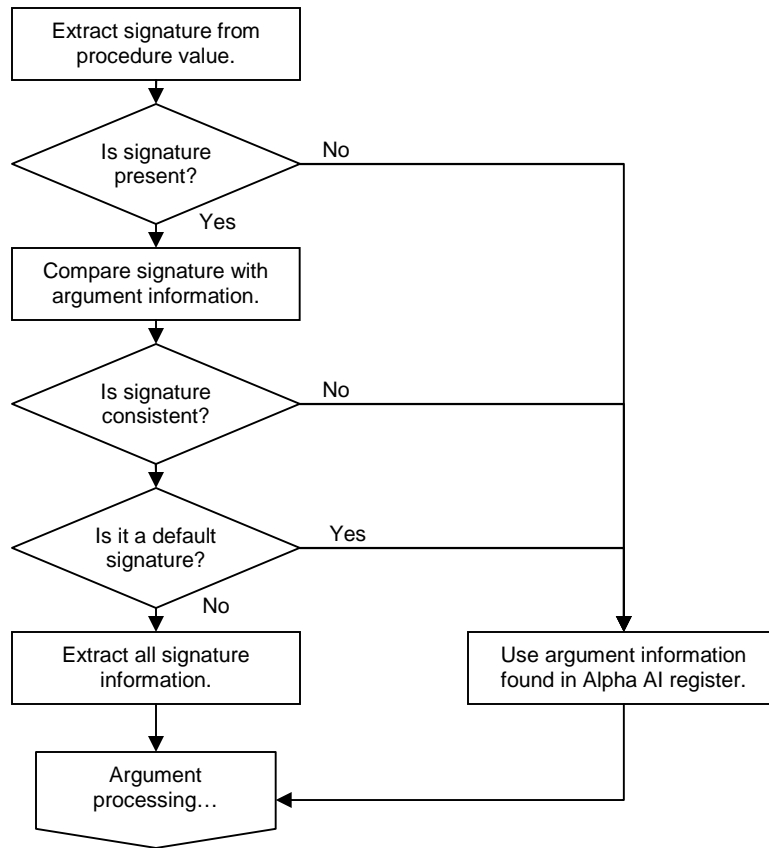
Switching Environments

The final stage in transferring control is the actual branch. As stated in previous sections, transferring from one foreign procedure to another (in the same processor environment) is nothing special. It is handled with a simple condition branch instruction. The real complexity starts when transferring control to another environment and trying to hide that from translated software.

Translated to Native

If a call address is determined to exist in a native image TIE$AXP_JUMP_TO transfers control by branching to TIE$ALPHA_TO_IPF. This routine is the final stepping stone for all translated code transferring control to the native OpenVMS I64 environment. The process begins by extracting the call signature information[4] from the callee's function descriptor. Figure 9 demonstrates the program logic that determines the final argument information that is used to process the argument list.

---

[4] See the *HP OpenVMS Calling Standard,* referenced in 'For more information' for further details.

Extract signature from procedure value.

Is signature present? — No

Yes

Compare signature with argument information.

Is signature consistent? — No

Is it a default signature? — Yes

No

Extract all signature information.

Use argument information found in Alpha AI register.

Argument processing…

Figure 9 Logic to Determine Argument Information

As Figure 9 shows, in the event that the signature is deemed unusable the fail over is to use argument information from the Alpha AI register (R25). The difference between the call signature and the argument information register is that a signature provides complete details of all arguments and return values. It is present purposefully for the translated environment and is usually only available in images that contain code that has been compiled with the /TIE qualifier. Argument information from the AI register, on the other hand, is available in all calls. However, it only details the format of the arguments present in registers.

The convention of using the argument information register as the default signature actually defies the calling standard. It states that when transferring control from the Alpha TIE to a native I64 procedure the default signature specifies that it should be assumed that all register arguments are RASE$K_RA_I32 and all memory arguments are MASE$K_MA_I32 (32 bit integers, sign extended to 64 bits). By using the argument information register, the default argument type is actually assumed to be AI$K_AR_I64 (64 bit integer). The default function result is also assumed to be PSIG$K_FR_I64 (64 bit integer).

Once all argument information has been extracted it is now necessary to prepare the arguments for the call to a native procedure. While both architectures employ similar

methods in argument passing, the differences are still far too great. Figure 10 shows the logic used to convert Alpha register arguments to their native I64 counterparts.
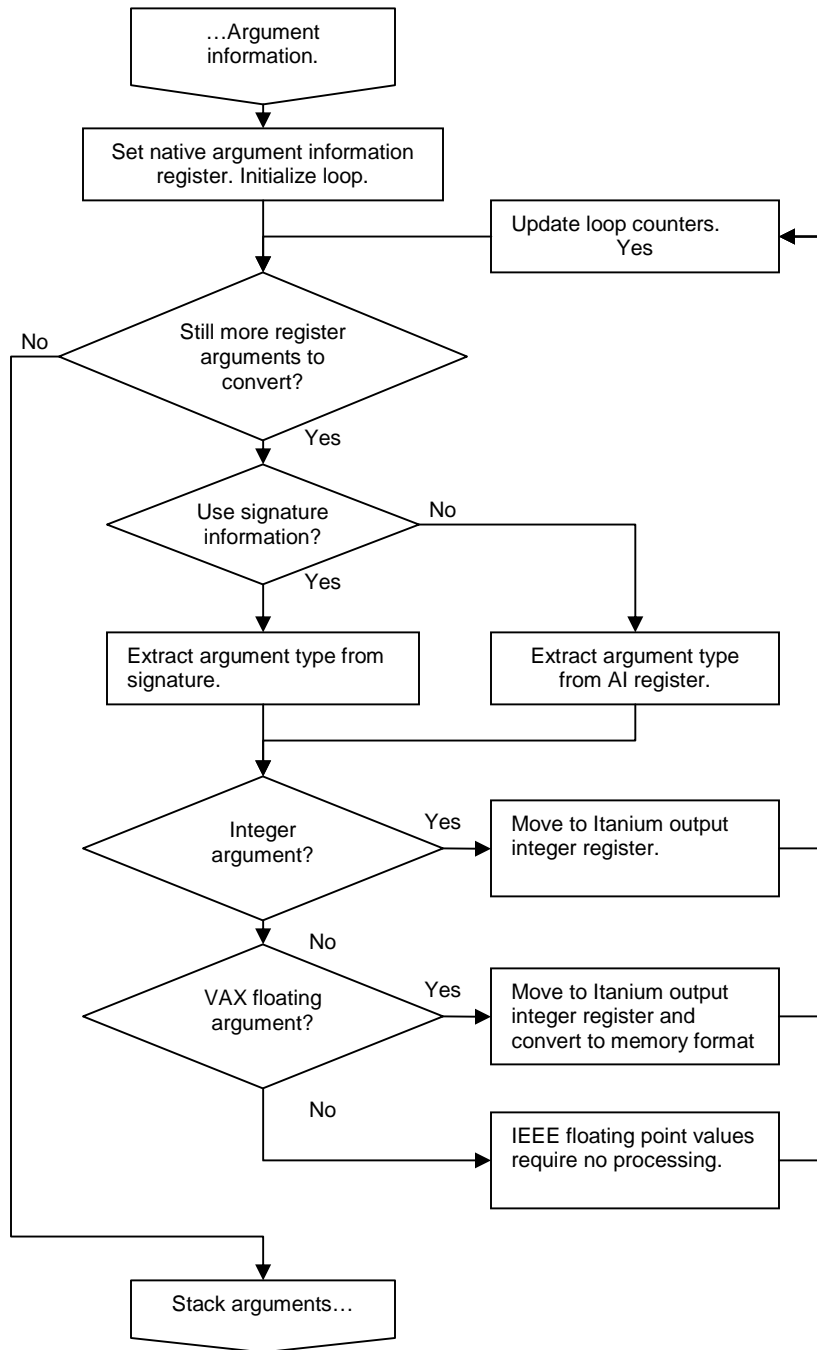
Figure 10 Register Argument Conversion Logic

Floating point arguments on the Alpha, like the Itanium, are passed in the floating point registers. However, the OpenVMS I64 calling standard requires that VAX floating point numbers (F_FLOAT, G_FLOAT and D_FLOAT) must be passed in general registers in memory format. TIE translated code stores all floating point numbers in register format

in native floating point registers. Therefore all any arguments must be moved to general registers and converted.

It is also a requirement of the Itanium architecture that a rotating region of output registers be used to pass arguments to the callee. Therefore it is necessary to copy all arguments in the Alpha registers R16-R21 (at least as many as the AI register dictates) into these output registers.

Lastly, there are less argument registers on the Alpha than there are on the Itanium. How this is handled is not shown in Figure 10. However, the process is quite simple. The arguments in slots 7 and 8 (stacked on the Alpha) are fetched and stored in their corresponding output registers. Again, it is necessary to consider the type of the argument to ensure it is stored in the correct register in the correct format. Moving these two arguments into register does have an advantage. The I64 calling standard requires that the caller leave a 16 byte scratch area before stacking any extra arguments passed to the callee. The stack locations formerly held by arguments 7 and 8 can then be used for this region preventing the need to re-shuffle the remaining stacked arguments. For calls with less than 8 arguments this scratch region is allocated by adjusting the stack pointer.

Once the arguments have been prepared the Alpha procedure descriptor is examined to determine the exception mode. Depending on the value in PDSC$V_EXCEPTION_MODE the call may pass through a wrapper that configures the floating point status application register (as.fpsr).

The final step before making the actual call is to move the contents of the Alpha registers R0 and R1 into their corresponding Itanium register, R8 and R9. Both of these registers can be used by compilers to pass information between calls. R1 (R9 on the Itanium) is often used to pass a pointer to the callers automatic storage to support uplevel references.

On returning from the native routine it is necessary to go through the process of converting the return value to an acceptable format. For return values the Itanium register R8 is copied into the Alpha register R0. For return values specified as PSIG$K_FR_D64, this also means that the contents of Itanium register R9 are moved to Alpha register R9. For all return values specified as VAX floating point they are converted back to register format and moved from the Itanium register F8 to Alpha register F0. For all complex values, the contents of Itanium register F9 is also moved in to Alpha register R1. For IEEE values there is no conversion. The contents of the Itanium registers F8 and F9 are copied to their Alpha equivalents. The register F9 is only copied in the case of a complex result.

At this point TIE$ALPHA_TO_IPF has completed the native call. The only thing left to do is to branch to the Alpha return address. This is done by branching to TIE$AXP_JUMP_TO.

Native to Translated

Although branching from native code into TIE might not strictly be emulation of a JSR instruction, it certainly needs to appear that way to the TIE. All transfers from native to translated code should be done through the routine OTS$CALL_PROC/TIE$NATIVE_TO_TRANSLATED[5]. While this routine is largely a wrapper, it is the interface used by all OpenVMS compilers. To call a native routine without using a compiler requires the call to be coded in Itanium assembly.

To begin, OTS$CALL_PROC calls TIE$NATIVE_TO_TRANSLATED which determines the environment it needs to transfer control to using the routine TIE$PROC_KIND5. For calls destined for the Alpha environment this means a branch to TIE$IPF_TO_ALPHA (for the VAX environment it is TIE$IPF_TO_VAX). This routine is responsible for the translation of all arguments and return values when going from native to translated code and back again. It can be thought of as the reverse of TIE$ALPHA_TO_IPF.

TIE$IPF_TO_ALPHA begins in much the same way as TIE$ALPHA_TO_IPF. It fetches the procedure signature and verifies it. If the signature is not found or invalid the argument information is retrieved from the Itanium argument information register (R25). The next step is to translate all the arguments as required. In the case of IEEE floating point and integer values there is no change. VAX floating point numbers are converted to register format and stored in the relevant floating point register. The 16 byte scratch region is used again. This time it holds any values found in the last two general register argument slots.

Once all arguments and the stack are prepared some Alpha registers are configured. The Alpha frame pointer is set and the Alpha return address is set to a special address that begins the process of converting return values. The contents of the Itanium registers R8 and R9 are also copied into the Alpha registers R0 and R1. The execution of the Alpha routine is then started by branching to TIE$AXP_JUMP_TO.

On return from the Alpha routine the return values are converted back to meet the requirements of the native OpenVMS I64 environment. This is done in much the same way as TIE$ALPHA_TO_IPF. The final action of TIE$IPF_TO_ALPHA is to branch back to the caller.

Routine Reference

The following are some quick notes on the calling details of the publicly defined TIE services mentioned in this article.

---

[5] See the section 'Routine Reference'.

TIE$PROC_KIND

This routine determines the 'kind' of a procedure address. This routine is used by OTS$CALL_PROC/TIE$NATIVE_TO_TRANSLATED to determine if it should transfer control to the TIE or not.

Parameters:

ofd.rq.v   Address of an official function descriptor. For native images this argument is the address of a function descriptor. In the case of translated Alpha images this is the address of a procedure descriptor. Lastly, for translated VAX images this is a pointer to the original VAX code.

Completion Codes:

The following table describes possible return values of this function. The symbolic names are found in [IA64_TIE]TIE_DEFS.H and not publicly available.

Value	Symbolic Name	Description
0	TIE$K_PROC_NATIVE	Address points to a native function descriptor.
1	TIE$K_PROC_ALPHA	Address points to an Alpha procedure descriptor.
2	TIE$K_PROC_VAX	Address points to a VAX procedure.

In the event that the function descriptor is not readable this routine will signal TIE$_WRONG_PV.

OTS$CALL_PROC/TIE$NATIVE_TO_TRANSLATED

The routine OTS$CALL_PROC is used by all OpenVMS compilers when calling external routines and the /TIE qualifier is active. In the event that the call is user mode and the TIE is active (CTL$GQ_TIE_SYMVECT) is non-zero) then OTS$CALL_PROC transfers control to TIE$NATIVE_TO_TRANSLATED (also known as TIE$CALL_PROC).

Parameters:

p1, …pn   All arguments are passed as if directly calling the routine in question. Floating point arguments are all passed in f16-f23. All other arguments are passed in r32-r39.

sig.rr.r   Address of the call argument signature block. This provides details of the arguments being passed. This is passed in register r17.

ofd.rr.r   The destination function descriptor. This is passed in register r18.

ai.rq.v   Argument information. This is passed in through register r25.

ra.ra.v   Return address. This is passed in the register b0.

In the case of calling a translated routine, registers r8 and r9 (up level environment value) are copied into R0 and R1 of the translated environment. They are then copied back on return.

Completion Codes:
    Only the completion codes of the routine being indirectly called.

Relevant Sources

The following list comprises the source modules from the OpenVMS I64 source tree that are discussed in this article.

- [IA64_TIE]AUX_CALLING_STANDARD.S
- [IA64_TIE]TIEDATA.H
- [IA64_TIE]TIE_CONT.C
- [IA64_TIE]TIE_CONT.H
- [IA64_TIE]TIE_DEFS.H
- [IA64_TIE]TIE_IMGSUP.C
- [IA64_TIE]TIE_JACKETS.S
- [IA64_TIE]TIE_JUMPS.S
- [LIBOTS]OTS$CALL_PROC_IA64.IAS
- [STARLET]FDSCDEF.SDL
- [STARLET]PDSCDEF.SDL
- [STARLET]PSIGDEF.SDL

For more information

Tim Sneddon can be contacted via email at tim.sneddon@bigpond.com.

For further information regarding the OpenVMS Alpha TIE and the VEST binary translation tools, please see the following:

- Goldenburg, R., Saravanan, S., Duma, D. *OpenVMS Alpha Internals and Data Structures: Scheduling and Process Control: V7.0* (1997) Digital Press ISBN: 978-1555581565

- Sites, R. L., Chernoff, A., Kirk, M. B., Marks, M. P., Robinson, S. G. *Binary Translation, Digital Technical Journal Vol. 4 No.4* (1992)
    - o http://www.dtjcd.vmsresource.org.uk/pdfs/dtj_v04-04_1992.pdf

- *HP OpenVMS Migration Software for VAX to Alpha Systems (OMSVA) Documentation*
    - o http://h71000.www7.hp.com/openvms/products/omsva/omsva.html

- For those readers with access to the OpenVMS Alpha source listings (or the source itself), the source for the TIE$SHARE.EXE run-time library can be found in the [TIE] facility.

For further information regarding the VAX architecture and instruction set, please see the following:

- *VAX Architecture Reference Manual* (1991) Digital Press ISBN: 978-1555580575

For further information regarding the OpenVMS I64 TIE and the AEST binary translation tool, please see the following:

- *HP OpenVMS Calling Standard*
    - http://h71000.www7.hp.com/doc/os83_index.html
- *HP OpenVMS Migration Software for HP AlphaServer Systems to HP Integrity Servers (MSAIS) Documentation*
    - http://h71000.www7.hp.com/openvms/products/omsva/omsais.html
- For those readers with access to the OpenVMS I64 source listings (or the source itself), the source for the TIE$SHARE.EXE run-time library can be found in the [IA64_TIE] facility.

For further information regarding the Alpha architecture, please see the following:

- Alpha Architecture Committee, Witek, R. T., *Alpha Architecture Reference Manual (Third Edition)* (1998) Digital Press ISBN: 978-1555582029

For further information regarding the Itanium, please see the following:

- Evans, J. S., Trimper, G. L. *Itanium Architecture for Programmers: Understanding 64-bit Processors and EPIC Principles* (2003) Hewlett-Packard Books ISBN: 0-13-101372-6
- *Intel® Itanium® Architecture Software Developer's Manual* (2006) Intel Corporation
    - http://www.intel.com/design/itanium/manuals/iiasdmanual.htm

For further information on the Fowler Noll Vo hashing algorithm, please see the following:

- Wikipedia, *Fowler Noll Vo hash*
    - http://en.wikipedia.org/wiki/Fowler_Noll_Vo_hash
- Noll, L. C., *FNV hash*
    - http://isthe.com/chongo/tech/comp/fnv/

OpenVMS Technical Journal V13

Trouble-shooting iCAP (Instant Capacity) on OpenVMS

Abu Sarkar

Executive Summary

This white paper outlines how HP OpenVMS is embracing the utility pricing solutions like iCAP/TiCAP/GiCAP and provides information on how to address issues in configuring iCAP and dependent products.

The following topics are covered:

- An introduction to the utility pricing solutions offered from HP

- Configuration of iCAP and Dependent software on HP Integrity servers

- Troubleshooting

  - Quick Check

  - ICAP_SERVER does not start on reboot of a partition in a complex

  - iCAP command does not work in a mixed environment due to HP-UX OS upgrade

  - HTTP Error (500 Internal Server Error)

Introduction to the Utility Pricing Solutions offered from HP

HP Instant Capacity software provides the ability to instantly increase or decrease computing capacity on partionable HP enterprise servers. The Instant Capacity software provides the means to:

• Increase or decrease (load balance) system processing capacity (icapmodify command).

• View status and configuration of the system components (icapstatus command).

• Administer system identification and notification information (icapmodify command).

• If configured, send system asset reports through encrypted email to HP (ICAP_SERVER process on OpenVMS).

• Send configuration change notification, through encrypted email, to the specified system contact.

• Monitor and report system compliance (ICAP_SERVER process on OpenVMS).

For the OpenVMS equivalents of these commands, see [DCL ICAP Commands](#) in the [Special OpenVMS-Specific Features and Considerations](#) section.

Note: HP Instant Capacity for HP 9000 and HP Integrity Servers, also known as Instant Capacity or iCAP, was known in earlier versions as Instant Capacity on Demand, or iCOD. Although the commands, warning messages and error messages refer to the software as iCAP, some internal files might still refer to iCOD.

With Instant Capacity, you initially purchase an HP enterprise server with a specified amount of active processing capacity, and a specified amount of inactive processing capacity. This amount can vary based on your sales contract with HP.

Processing capacity consists of the system components:

• *Processors* containing *cores*

• *Cell boards*

For each type of component, the number of components that can be active is equal to the number of usage rights applied to the complex for that type of component. Components that are purchased with a part number identifying them as "Instant Capacity" and without the label "Right to Use" come without usage rights. Components that are not labeled "Instant Capacity" implicitly include usage rights that can be applied to any component of that type that is installed on the complex.

Prior to activation of an inactive component, you must obtain additional **usage rights**. The fundamental method is to purchase usage rights by purchasing the appropriate Instant Capacity products that include the label "Right to Use (RTU)". HP then supplies an **RTU codeword**. When the codeword is applied to the HP enterprise server, the inactive component can be activated.

Additional methods for activating components for which usage rights have not been purchased include:

- If a server is a member of a *Global Instant Capacity (GiCAP)* group, and if extra capacity is available from other members of the group, capacity can be "borrowed" from another member of the group.

- You can purchase additional temporary capacity and apply the *Temporary Instant Capacity (TiCAP)* codeword in order to activate one or more cores temporarily. If a server is a member of a GiCAP group, temporary capacity can be shared among members of the group.

- You can temporarily activate one or more inactive cores using the *Instant Access Capacity (IAC)* provided with the initial purchase of the Instant Capacity component. Instant Access Capacity is the same as TiCAP except it is automatically provided with an Instant Capacity component and cannot be purchased separately. It provides an immediate buffer of temporary capacity in case extra capacity is needed before there is time to purchase an RTU or a TiCAP codeword.

Note: It is always a good idea to keep some quantity of temporary capacity in reserve. Purchase of codewords may take one or more days, so having a buffer of temporary capacity allows you to avoid delays in activation of additional cores. Instant Access Capacity provides this buffer initially, but as that capacity is depleted, ongoing purchases of additional temporary capacity are recommended to replenish it. *Global Instant Capacity* features, including the use of the *icapmanage* command, are not supported on OpenVMS.

Instant Capacity Software in Virtual Environment

Instant Capacity must be run on a partitionable system. In an *Integrity Virtual Machine (VM)* environment, Instant Capacity software provides meaningful functionality only on the *VM Host* system; it does not run on a *virtual machine* (also known as a "guest").

Utility Pricing Solutions Portal

The Utility Pricing Solutions (or Instant Capacity) portal is located at the HP web site: http://www.hp.com/go/icap/portal
After you purchase a component without usage rights, HP sends you a letter containing instructions on how to obtain an RTU codeword from the Utility Pricing Solutions portal.

Instant Capacity Administration System

If asset reporting is configured, the ICAP_SERVER process sends asset reports, in the form of encrypted email messages, to the Instant Capacity Administration System, which saves information in the Instant Capacity database.

Instant Capacity Database

The Instant Capacity database is a repository on an HP (internal) corporate server that tracks system resources and provides the information for codeword generation. Note that

this database should not be confused with a Global Instant Capacity database, which is created on a customer Group Manager system.
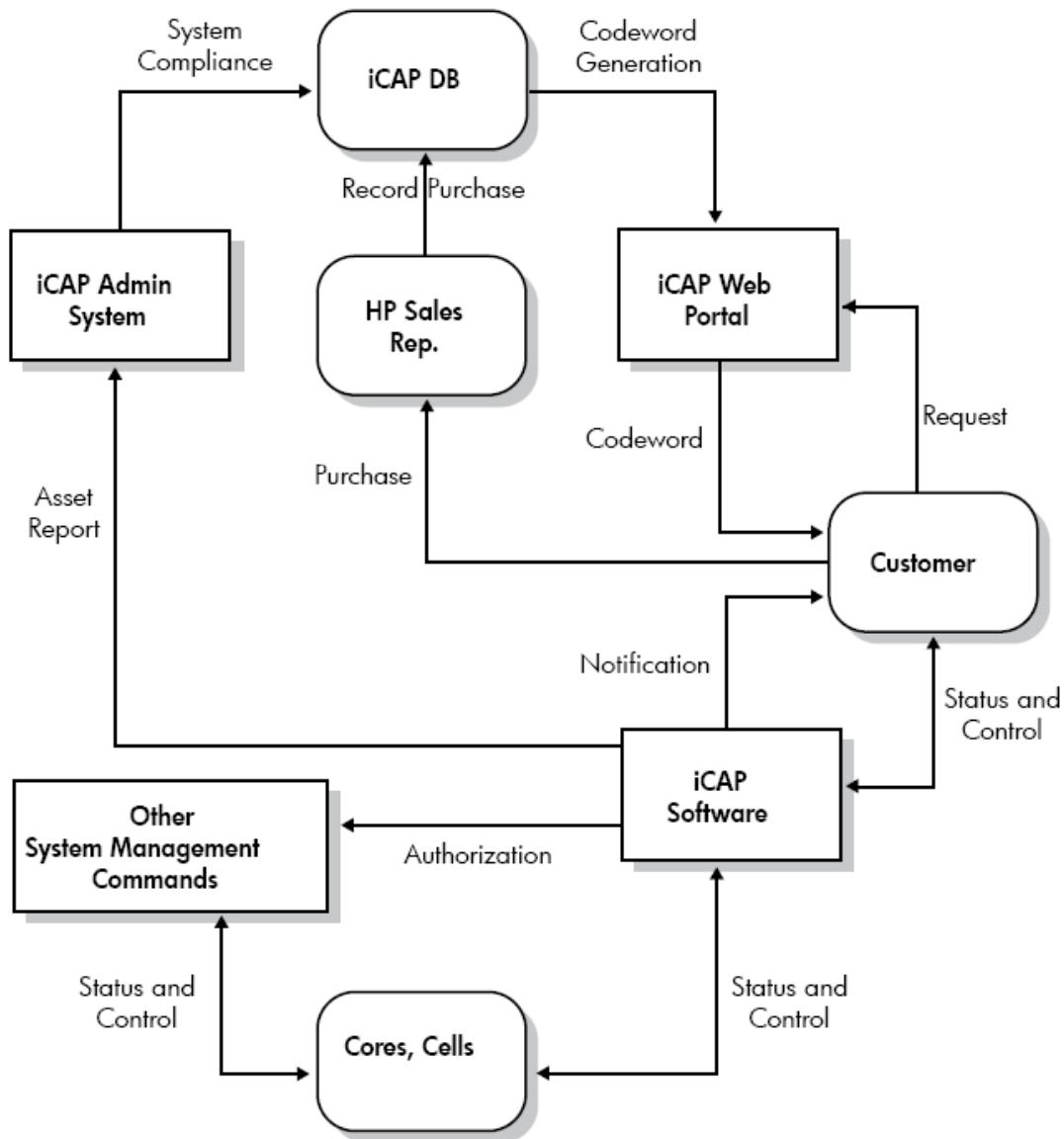

Fig: Instant Capacity System Elements

Special OpenVMS-Specific Features and Considerations

Core Activation and Deactivation

The ICAP command or the corresponding HP-UX foreign commands must be used on OpenVMS systems when stopping and starting CPUs in complexes containing iCAP components. To start cores on OpenVMS, use the ICAP ACTIVATE/CPU= command.

To stop cores on OpenVMS, use the ICAP DEACTIVATE/CPU= command. HP recommends not to use START /CPU and STOP /CPU command or the corresponding system services in order to start or stop processor resources in an iCAP complex. When you enter the START /CPU command on an OpenVMS system in a complex containing iCAP resources, the ICAP_SERVER process validates that the start operation does not take the complex out of compliance. When you enter the STOP /CPU command, the CPU might restart at a later time if the count of intended active cores on the system is greater than the actual active cores. Using the START /CPU command can result in unintended consequences, such as an unexpected usage of temporary capacity or the deactivation of cores on the system or on another system in the complex. Using the STOP /CPU command can result in an unexpected restart of the core or the unexpected start of a core in another system in the complex.

Email Considerations

The iCAP software requires that SMTP mail be configured on the OpenVMS system to send email to the system contact. For more information about setting up SMTP mail, see your IP provider's documentation.

Time zone Considerations

On OpenVMS systems, the ICAP_SERVER process performs routine Instant Capacity software tasks on a daily basis. A partition's local time zone setting affects what time zone the ICAP_SERVER process uses for the timing of these tasks. Be sure that the time zone is set properly to ensure synchronization among the partitions.

On OpenVMS systems, the ICAP_SERVER uses the time zone settings defined by the `SYS$STARTUP:TDF$UTC_STARTUP.COM` file.

To view the time zone settings, enter the command
$ @sys$manager:utc$time_setup "show"

Enter the command
$ @sys$manager:utc$time_setup

and follow the menu instructions to modify the time zone setting for the iCAP partition.

Restrictions

- Instant Capacity software on OpenVMS Version 8.3-1H1 does not support HP virtual partitioning (vPars) since OpenVMS itself doesn't support vPars.
- Global Instant Capacity features, including the use of the icapmanage command, are not supported on OpenVMS.
- Instant Capacity on OpenVMS does not support internationalization. Only English language support is provided.
- LPMC and HPMC are not available on OpenVMS systems.

Removing Instant Capacity Software

To participate in the Instant Capacity version 9.x program, you must comply with the
following conditions of the HP Utility Pricing Solutions program:
• Maintain the HP Instant Capacity software on each HP-UX or OpenVMS partition in
the system. The Instant Capacity software is a nonintrusive, low-overhead software
module that resides on the partition.

• Migrate to later Instant Capacity software versions as they become available. For
specifics about your individual program requirements, see the Utility Pricing Solutions
contract from HP or your authorized channel partner.

NOTE: HP recommends that you do not remove the Instant Capacity software.
Participants in the Utility Pricing Solutions program who do not meet these requirements
may be in breach of contract. This can result in unnecessary expense for both the program
participant and HP.

Configuration of iCAP and Dependent software on HP Integrity servers

iCAP Product Versions and Supported Platforms

The table lists the current versions of Instant Capacity and the platforms supported for
each version.

Software and Version	Operating System Version	Supported Hardware Platforms	Notes
iCAP 8.1 (BA484AA)	HP OpenVMS Version 8.3-1H1	HP Integrity servers: • Superdome • rx8640 • rx8620 • rx7640 • rx7620	Available on: • OpenVMS Version 8.3-1H1 Operating System media

Special Requirements

Installation disk of all the products (including the dependency products & patches) should
be ODS-5 formatted since UNIX-style file names are used in some cases, which is not
supported on ODS-2 formatted disks.

Dependency Products

• HP TCP/IP Services of OpenVMS (ECO2 is required for TCP/IP Services Versions
  5.5 and 5.6)
• HP SSL V1.3
• HP WBEM Services V2.61

Please do maintain the order of installation of the dependency products as mentioned above.

Required Patches

*   HP I64VMS VMS831H1I_PCSI V1.0,
*   HP I64VMS VMS831H1I_UPDATE V2.0.

Note: Please do maintain the order of installation of the patches as mentioned above since the later is dependent on the previous one. It is recommended to upgrade your system with the latest SYS and UPDATE patches. Update patches make functional changes to the system and requires reboot in order to have the images in the kit to fully take effect. The patches could be downloaded from the hp ITRC site.

Compatibility Matrix

Operating System/Patch version	WBEM Services version	iCAP version
HP I64VMS OPENVMS V8.3-1H1	HP I64VMS WBEMCIM V2.61-A070728	iCAP B.08.02
VMS831H1_ICAP-V0200 ECO kit (releasing shortly)	HP I64VMS WBEMCIM V2.91-A090219	iCAP B.08.02

Note that not all the WBEM Services versions are compliant with the iCAP software.

Installation & Configuration of WBEM Services for OpenVMS

WBEM Services for OpenVMS is installed automatically with OpenVMS. As with other similar products, an OpenVMS upgrade does not automatically include WBEM Services for OpenVMS if it is not already installed on the target system disk. In this case, you must install the product separately using the PCSI PRODUCT INSTALL command. You must configure WBEM Services for OpenVMS to obtain the services provided by HP SIM (Version 5.2 or later) and products such as Instant Capacity, and gWLM. To provide services over the network, HP recommends using TCP/IP Services for OpenVMS and SSL (for security purposes).

Before configuring WBEM Services for OpenVMS, configure TCP/IP Services for OpenVMS. For information about configuring TCP/IP Services for OpenVMS, see Section 7.7.5 (page 122) of
HP OpenVMS Version 8.3-1H1 for Integrity Servers Upgrade and Installation Manual.

To configure WBEM Services for OpenVMS on a system on which WBEM Services for OpenVMS has never been installed and configured, follow the steps described in Section 4.1. If you are configuring the product on a system on which it has been configured previously, see Section 4.2.

For more information about HP WBEM products, see the following website:
http://www.hp.com/go/wbem.

Configuring WBEM Services for OpenVMS (Where Not   Configured Previously)

To configure WBEM Services for OpenVMS on a system for the first time, follow these steps:

1. Enter the following command
   ```
   $ RUN SYS$SYSROOT:[WBEM_SERVICES]WBEM_SERVICES$CONFIG
   ```

This command invokes the utility that configures and initializes the environment for WBEM Services for OpenVMS.

2. After displaying the initial configuration utility banner, the utility informs you where it will store the configuration files and repository and asks if you want to change the location.

The configuration files and repository will be placed in the following location: SYS$SPECIFIC:[WBEM_Services].

```
Do you want to change this location (Yes/No) [No]?:
```

Note: The repository, a compiled version of the Common Information Model (CIM) class schema, requires an ODS-5 formatted disk (the repository uses UNIX-style file names, which are not supported on ODS-2 formatted disks). If the default location is on an ODS-2 formatted disk, you must change the location to an ODS-5 disk.

When you accept the default location, the utility informs you that all configuration questions have been answered and asks whether you want to continue, as shown in the following example. If you choose to continue, the utility creates the CIMSERVER repository tree in the location indicated earlier. The CIMSERVER is the WBEM Services for OpenVMS process that runs on the system to support certain applications. It also creates the following command files:

```
SYS$STARTUP:WBEM_Services$Startup.com
SYS$STARTUP:WBEM_Services$Shutdown.com
SYS$SYSROOT:[WBEM_SERVICES]WBEM_Services$Define_Commands.com
```

The SYS$STARTUP:WBEM_Services$Startup.com file defines system logicals for the WBEM Services for OpenVMS environment.

All configuration questions have been answered.

```
Do you want to continue (Yes/No) [YES]?:
%WBEMCONFIG-I-CREREPBEGIN, Create Repository Begins...
%WBEMCONFIG-I-CREREPCOMPLETE, Create Repository Complete.
```

```
This utility creates:
    SYS$STARTUP:WBEM_Services$Startup.com
which should be added to SYS$STARTUP:SYSTARTUP_VMS.COM.

This utility creates:
    SYS$STARTUP:WBEM_Services$Shutdown.com
which should be added to SYS$STARTUP:SYSHUTDWN.COM.

This utility creates:
    SYS$SYSROOT:[wbem_services]WBEM_Services$Define_Commands.com
which users who use this product can add to their login.com.
```

3. The utility asks whether to start the CIMSERVER:

```
Do you want to start the CIMServer now (Yes/No) [Yes]?:
```

CIMSERVER must be running so that your system can use such applications as Instant Capacity, and gWLM. You can start CIMSERVER now, or you can perform other post-installation or post-upgrade tasks first and then start CIMSERVER. If you choose to start CIMSERVER now, the utility displays the progress and operating system information, as in the following example:

```
%RUN-S-PROC_ID, identification of created process is 21A00599
%WBEMCIM-I-STARTUPWAIT, Waiting for CIMServer to start... 120 seconds
remaining.
%WBEMCIM-S-CSSTARTED, CIMServer successfully started.
OperatingSystem Information
  Host: boston.hp.com
  Name: OpenVMS
  Version: V8.3-1H1
  UserLicense: Unlimited user license
  Number of Users: 1 users
  Number of Processes: 29 processes
  OSCapability: 64 bit
  LastBootTime: Jul 31, 2007  10:52:55 (-0400)
  LocalDateTime: Aug 3, 2007  10:14:58 (-0400)
  SystemUpTime: 256923 seconds = 2 days, 23 hrs, 22 mins, 3 secs
```

4. To ensure that CIMServer starts automatically at each reboot, add the following line to the
    SYS$MANAGER:SYSTARTUP_VMS.COM file:
        $ @SYS$STARTUP:WBEM_Services$Startup.com

To have CIMServer shut down automatically with the operating system, add the following line to the SYS$MANAGER:SYSSTARTUP:SYSHUTDWN.COM file:

```
$ @SYS$STARTUP:WBEM_Services$Shutdown.com
```

All users who use this product should also add the following line to their LOGIN.COM file:
```
$ @SYS$STARTUP:WBEM_Services$Define_Commands.com
```

5. In an OpenVMS Cluster, each member that runs WBEM Services for OpenVMS needs its own repository. Therefore, you must perform the WBEM Services for OpenVMS configuration procedure on each of those cluster members.

Configuring WBEM Services for OpenVMS (Where Configured Previously)

To configure WBEM Services for OpenVMS on a system where it has been configured previously, follow these steps:

1. Enter the following command
```
$  RUN SYS$SYSROOT:[WBEM_SERVICES]WBEM_SERVICES$CONFIG
```

This command starts the utility that configures and initializes the environment for WBEM Services for OpenVMS.

If the WBEM Services for OpenVMS product (Version 2.0) available with OpenVMS I64 Version 8.3 is already configured on your system, the following error message and the recommended remedial actions appear:

```
%WBEMCONFIG-E-SYSCOMMONLOGICAL, WBEM_VAR can no longer be defined to
point to a location in SYS$COMMON. The repository files in WBEM_VAR
should not be shared with other cluster members.
```

Follow these manual steps to move the repository out of the SYS$COMMON area and complete the post installation configuration tasks:

- Delete the sys$common:[WBEM_Services.var...] directory tree.
- Deassign the WBEM_VAR system logical.
- Run this procedure again.

Perform the recommended steps, as in the following example:

```
$  DELETE SYS$COMMON:[WBEM_SERVICES.VAR]*.*;*
$  DELETE SYS$COMMON:[WBEM_SERVICES]VAR.DIR;*
$  DEASSIGN/SYS WBEM_VAR
$  RUN SYS$SYSROOT:[WBEM_SERVICES]WBEM_SERVICES$CONFIG
```

After you start the configuration procedure, go to Section A and follow the steps described there, starting with step 2.

2. After displaying the initial configuration utility banner, the utility informs you where it will store the configuration files and repository and asks if you want to change the location.

The configuration files and repository will be placed in the following location:

```
SYS$SPECIFIC:[WBEM_Services].

Do you want to change this location (Yes/No) [No]?:
```

The repository is a compiled version of the CIM class schema. This example assumes you accept the current location.

3. As shown in the following example, the utility informs you that all configuration questions have been answered and asks whether you want to continue.

If the utility determines that the repository schema has not changed, the utility informs you and continues. The utility does not need to upgrade the repository.

If the utility determines that the current repository needs upgrading, or if the utility does not find a repository (perhaps WBEM Services for OpenVMS had been installed but not configured), the utility displays a message informing you that the repository will be upgraded or created and that this will take 10 to 15 minutes depending on your processor and disk I/O speed. In the following example, the utility needs to create the repository tree.

The utility also creates the `SYS$STARTUP:WBEM_Services$Startup.com`, `SYS$STARTUP:WBEM_Services$Shutdown.com`, and `SYS$SYSROOT:[WBEM_SERVICES]WBEM_Services$Define_Commands.com` command files. The `SYS$STARTUP:WBEM_Services$Startup.com` file defines system logicals for the WBEM Services for OpenVMS environment.

```
All configuration questions have been answered.

Do you want to continue (Yes/No) [Yes]?:

%WBEMCONFIG-I-CREREPBEGIN, Create Repository Begins...
%WBEMCONFIG-I-CREREPCOMPLETE, Create Repository Complete.
This utility creates:
    SYS$STARTUP:WBEM_Services$Startup.com
which should be added to SYS$STARTUP:SYSTARTUP_VMS.COM.

This utility creates:
    SYS$STARTUP:WBEM_Services$Shutdown.com
which should be added to SYS$STARTUP:SYSHUTDWN.COM.

This utility creates:
    SYS$SYSROOT:[wbem_services]WBEM_Services$Define_Commands.com
which users who use this product can add to their login.com.
```

4. The utility now asks you whether to start the CIMSERVER:

```
Do you want to start the CIMServer now (Y/N) {Y}?:
```

CIMSERVER must be running so that your system can use such applications as Instant Capacity, Pay per use, and gWLM. You can start CIMSERVER now, or you can perform other post-installation or post-upgrade tasks first and then start CIMSERVER. If you

choose to start CIMSERVER now, the utility displays the progress and operating system information, as in the following example:

```
%RUN-S-PROC_ID, identification of created process is 21A00599
%WBEMCIM-I-STARTUPWAIT, Waiting for CIMServer to start... 120 seconds
remaining.
%WBEMCIM-S-CSSTARTED, CIMServer successfully started.
OperatingSystem Information
  Host: boston.hp.com
  Name: OpenVMS
  Version: V8.3-1H1
  UserLicense: Unlimited user license
  Number of Users: 1 users
  Number of Processes: 29 processes
  OSCapability: 64 bit
  LastBootTime: Jul 31, 2007  10:52:55 (-0400)
  LocalDateTime: Aug 3, 2007  10:14:58 (-0400)
  SystemUpTime: 256923 seconds = 2 days, 23 hrs, 22 mins, 3 secs
```

5. To ensure that CIMSERVER starts automatically at each reboot, add the following line to the

```
SYS$MANAGER:SYSTARTUP_VMS.COM file:
$ @SYS$STARTUP:WBEM_Services$Startup.com
```

To have CIMServer shut down automatically with the operating system, add the following line to the `SYS$MANAGER:SYSSTARTUP:SYSHUTDWN.COM` file:

```
$ @SYS$STARTUP:WBEM_Services$Shutdown.com
```

All users who use this product should also add the following line to their `LOGIN.COM` file:

```
$ @SYS$STARTUP:WBEM_Services$Define_Commands.com
```

6. In an OpenVMS Cluster, each member that will run WBEM Services for OpenVMS needs its own repository. Therefore, you must perform the WBEM Services for OpenVMS configuration procedure on each of those cluster members.

NOTE: HP recommends that you do not remove the WBEM Services for OpenVMS product even if you do not have a need for it. If you attempt to use the PRODUCT REMOVE command to remove this product, you might see a message similar to the following. This message is automatically displayed for any product that is required with OpenVMS. The consequences of removing WBEM Services for OpenVMS might not be as severe as implied by the message unless other software is using the product on your server.

```
%PCSI-E-HRDREF, product HP I64VMS WBEMCIM V2.61 is referenced by HP
I64VMS OPENVMS V8.3-1H1

   The two products listed above are tightly bound by a software
dependency.
   If you override the recommendation to terminate the operation, the
```

```
    referenced product will be removed, but the referencing product will
have an unsatisfied software dependency and may no longer function
correctly.
    Please review the referencing product's documentation on
requirements.

    Answer YES to the following question to terminate the PRODUCT
command.
    However, if you are sure you want to remove the referenced product
then answer NO to continue the operation.

 Terminating is strongly recommended. Do you want to terminate? [YES]
```

Installation & Configuration of Instant Capacity (iCAP) on OpenVMS Systems

Post installation & configuration of OpenVMS V8.3-1H1, WBEMCIM V2.61-A070728 kits, install the following update patches with /SAVE_RECOVERY_DATA qualifier (so that in case of the patches not functioning properly, we can revert to the previous state) in the order mentioned below and reboot the system.

```
HP I64VMS VMS831H1I_PCSI V1.0
HP I64VMS VMS831H1I_UPDATE V2.0
```

The above-mentioned OpenVMS V8.3-1H1 patches need to be installed for the following fixes:
- nPAR Provider fix for dynamic profile write error;
- C Runtime Library channel leak fix;
- iCAP and nPAR fixes to support Hyperthreading on Montecito.

NOTE: Update patches make functional changes to the system and requires reboot in order to have the images in the kit to fully take effect.

```
    This kit will make functional changes to your system.
    Before installing this kit you should make a backup
    copy of your system disk.  If you do not make a copy
    of your system disk you will not be able to restore
    your system to a pre-kit installation state.


    Do you want to continue? [YES]

    Installing this patch kit requires a reboot.


    Hewlett Packard strongly recommends that you reboot your
    system immediately after installation  of  this kit.  The
    images in this kit will not fully take effect until the
    system is rebooted.  However, if you do not re-boot
    immediately after kit installation, the system may become
    unstable and may not function as expected.
```

```
If you have other nodes in your VMS cluster, they must also
be rebooted in order to make use of the new image(s).  If it
is not possible or convenient to reboot the entire cluster
at this time, a rolling re-boot (kit installation and reboot
on one node at a time) may be performed.
```

Before configuring the iCAP software, start the CIMSERVER process by executing the following command procedure:

```
$ @SYS$STARTUP:WBEM_Services$Startup.com
```

Define the WBEM Services logical with the following command:

```
$ @SYS$COMMON:[WBEM_SERVICES]WBEM_Services$Define_Commands.com
```

Ensure that the CIM Server is running and verify the list of Providers installed by entering the following command:

```
$ CIMPROVIDER -L -S
```

An output similar to the following is displayed:

```
MODULE                          STATUS
OperatingSystemModule           OK
ComputerSystemModule            OK
ProcessModule                   OK
ProcessorProviderModule         OK
IPProviderModule                OK
```

Execute the following command procedure in order to configure the iCAP software:

```
$ @SYS$MANAGER:ICAP$CONFIG.COM
```

Note that, this procedure will stop and restart the CIMSERVER process during the configuration. The procedure will enquire if you want to configure GiCAP (Answer **NO** since GiCAP is not supported on OpenVMS V8.3-1H1), system-contact's e-mail address, configuration change notification and start the iCAP software.

A sample configuration output is provided below for your reference:

```
$ @SYS$MANAGER:ICAP$CONFIG.COM

          hp OpenVMS Industry Standard 64

    Instant Capacity on Demand (iCAP) configuration utility


    **************** W A R N I N G ****************

    This procedure stops and restarts the CIMSERVER
    process. ALL WBEM provider modules will be
```

unavailable for a short period of time during the
        configuration.

        **************** W A R N I N G ****************

%DCL-I-SUPERSEDE, previous value of SRC_MOF$ has been superseded
%ICAP-I-CHECK, Checking for iCAP configuration requirements
%ICAP-I-CHEDONE, Check Done requirements OK

Are you satisfied with the backup of your WBEMCIM repository (Yes/No)?:
Y
%ICAP-I-UNREGNPAR, Unregistering at 30-OCT-2008 08:04:57.32
Disabling provider module...
Provider module disabled successfully.
Deleting provider module...
Provider module deleted successfully.
Disabling provider module...
Provider module disabled successfully.
Deleting provider module...
Provider module deleted successfully.
Disabling provider module...
Provider module disabled successfully.
Deleting provider module...
Provider module deleted successfully.
Disabling provider module...
Provider module disabled successfully.
Deleting provider module...
Provider module deleted successfully.
%ICAP-I-UNREGDON, Unregistering iCAP/nPAR modules done at 30-OCT-2008
08:05:47.90
%ICAP-I-STOPCIM, Stopping the cimserver process.
%WBEMCIM-I-SHUTDOWN, Shutting down WBEM Services for OpenVMS......
%WBEMCIM-I-SHUTDOWNPROV, Shutting down WBEM Providers...
%WBEMPROVIDERS-I-SHUTDOWN, Info:Shutting down WBEMPROVIDERS.
%WBEMCIM-I-SHUTDOWNCS, Shutting down CIMServer.exe...
CIM Server stopped.
%WBEMCIM-I-CSEXITSTS, CIMServer.exe exit status=%X00000001.
%ICAP-I-RESCIM, Restarting the cimserver process.
%RUN-S-PROC_ID, identification of created process is 00000474
%WBEMCIM-I-STARTUPWAIT, Waiting for CIMServer to start... 120 seconds
remaining.
%WBEMCIM-S-CSSTARTED, CIMServer successfully started.
OperatingSystem Information
  Host: part0.ind.hp.com
  Name: OpenVMS
  Version: V8.3-1H1
  UserLicense: Unknown
  Number of Users: 1 users
  Number of Processes: 18 processes
  OSCapability: 64 bit
  LastBootTime: Oct 30, 2008 6:33:16 (00000)
  LocalDateTime: Oct 30, 2008 2:36:04 (00000)
  SystemUpTime: 4294953064 seconds = 49710 days, 2 hrs, 31 mins, 4 secs
%WBEMCIM-I-STARTPROV, Starting WBEM Providers...
%WBEMPROVIDERS-I-STARTING , Info:Starting WBEMPROVIDERS.
%WBEMPROVIDERS-I-WAIT, Info:Waiting for 1 Minute for the Inventory to
Initialize

```
%RUN-S-PROC_ID, identification of created process is 00000475
%iCAP-I-CRENAM, Creating root/cimv2/npar namespace
%ICAP-I-BEGUPDREP, Begin updating WBEMCIM repository at 30-OCT-2008
08:07:20.49


%ICAP-I-ENDUPDREP,   End updating WBEMCIM repository at 30-OCT-2008
08:08:44.58
%ICAP-I-REGNPAR, Registering iCAP Mofs at  30-OCT-2008 08:08:44.58


%ICAP-I-REGDone, Registering iCAP Mofs Done at 30-OCT-2008 08:09:24.46
Registering iCAP Command Language Definition file...
Command Language Definition file successfully registered
Enter (Y)es to configure this system with GiCAP support (N):
Would you like to set the System-Contact's E-mail Address? (Y/N): y
Enter the System-Contact's E-mail Address: xyz@hp.com

The contact e-mail address has been set to xyz@hp.com.

Would you like to turn configuration change notification on? (Y/N): y
Configuration change notification has been turned on.

Would you like to start the iCAP software now? (Y/N): y
%RUN-S-PROC_ID, identification of created process is 00000480

%ICAP-I-EXIT, Exiting ICAP configuration elapsed time 0:05:40.64
```

Check if all the iCAP related Providers are installed and running by entering the following command:

```
$ CIMPROVIDER -L -S
```

An output similar to the following is displayed:

```
MODULE                          STATUS
OperatingSystemModule           OK
ComputerSystemModule            OK
ProcessModule                   OK
ProcessorProviderModule         OK
IPProviderModule                OK
HP_NParProviderModule           OK
HP_iCODProviderModule           OK
HP_iCAPProviderModule           OK
HP_GiCAPProviderModule          OK
```

To verify that the Instant Capacity software is installed and configured, run the following OpenVMS commands:

```
$ @SYS$MANAGER:ICAP$CLI_UTILS.COM CONFIG_CHECK
$ show log ICAP$CONFIGURED
"ICAP$CONFIGURED" = "TRUE" (LNM$JOB_nnnnnnnn)
```

Troubleshooting

- Quick Check
- ICAP_SERVER doesn't start on reboot of a partition in a complex
- iCAP command doesn't work in a mixed environment due to HP-UX OS upgrade
- HTTP Error (500 Internal Server Error)

Quick Check

Check if the WBEM Services is running and all the iCAP related providers are configured with the following commands:

```
$ SHOW SYSTEM/PROCESS=CIMSERVER
```

An output similar to the following is displayed:

```
OpenVMS V8.3-1H1  on node PART0    25-FEB-2009 03:26:56.17   Uptime  12
01:34:27
  Pid    Process Name    State  Pri     I/O       CPU        Page flts
Pages
00000C42 CIMSERVER       HIB     10  1041866   0 00:07:00.87      7572
4259 M

$ @SYS$COMMON:[WBEM_SERVICES]WBEM_SERVICES$DEFINE_COMMANDS.COM
$ CIMPROVIDER -L -S
```

An output similar to the following is displayed:

```
MODULE                        STATUS
OperatingSystemModule         OK
ComputerSystemModule          OK
ProcessModule                 OK
ProcessorProviderModule       OK
IPProviderModule              OK
HP_NParProviderModule         OK    << needed by icap
HP_iCODProviderModule         OK    << needed by icap
HP_iCAPProviderModule         OK    << needed by icap
HP_GiCAPProviderModule        OK    << needed by icap (for GiCAP
functionality)
```

Verify that the Instant Capacity software is installed and configured, run the following OpenVMS commands:

```
$ SHOW SYSTEM/PROCESS=ICAP_SERVER
```

An output similar to the following is displayed:

```
OpenVMS V8.3-1H1  on node PART0    25-FEB-2009 03:28:44.19   Uptime  12
01:36:15
  Pid    Process Name    State  Pri     I/O       CPU        Page flts
Pages
```

```
00000471 ICAP_SERVER    HIB    10    465890   0 00:01:07.25     1408
1894
```

```
$ @SYS$MANAGER:ICAP$CLI_UTILS.COM CONFIG_CHECK
$ show log ICAP$CONFIGURED
"ICAP$CONFIGURED" = "TRUE" (LNM$JOB_nnnnnnnn)
```

ICAP_SERVER Does not Start on Reboot of a Partition in a Complex

ICAP_SERVER process should start automatically post reboot of a partition in a complex; it takes a few minutes, as it needs to wait for ERRFMT to write some data. Wait for a few minutes, even after that if ICAP_SERVER doesn't get started, then check if the HP TCP/IP & HP WBEMCIM services is configured properly and running. HP TCP/IP & HP WBEMCIM services should be running post reboot in order to run the ICAP_SERVER. Ensure that the following lines are appended to the SYS$STARTUP:SYSTARTUP_VMS.COM file in order get the HP TCP/IP & HP WBEMCIM services started automatically post reboot.

```
$ @SYS$STARTUP:TCPIP$STARTUP.COM
$ @SYS$STARTUP:WBEM_Services$Startup.com
```

iCAP Command Does not Work in a Mixed Environment due to HP-UX OS Upgrade

Customer might find some OPCOM messages like "failed to update the dynamic profile" displayed on the VMS partition for all iCAP commands after a HP-UX partition was upgraded. Prior to upgrading, customer is advised to read the documentation and if it states that all partitions must be upgraded, they should contact HP to validate if the HP-UX patch is compatible with the VMS version.

HTTP Error (500 Internal Server Error)

```
$ icap show status
ERROR:   The following low-level error occurred:

HTTP Error (500 Internal Server Error).
```

This error typically occurs when the CIMSERVER process has exhausted some resource. Pagefile quota and i/o channels are two resources that have had problems in the past. If this error is seen the CIMSERVER process should be examined with $ SHOW PROCESS /CONTINUOUS hit <Q> command. The i/o channels in use can be examined using SDA.

If CIMSERVER runs out of channel than a newer version of DECC$SHR.EXE is needed.

If the CIMSERVER process "Total number of channels" assigned is very close to the sysgen paramter CHANNELCNT, the CIMSERVER will loop or hang. As a workaround customer is advised to stop and restart the CIMSERVER.

CLI Support on OpenVMS

OpenVMS provides a CLI (command-line interface) to the Instant Capacity software. The HP-UX command syntax can be implemented using foreign command symbols. The DCL ICAP command provides DCL command support.

HP-UX Style Commands

The HP-UX command syntax can be used on OpenVMS systems by defining foreign command symbols to the iCAP images. Add the following three symbol declarations to your LOGIN.COM file or to the SYLOGIN file to define commands that use the HP-UX syntax:

```
$ icapmodify :== $ICAP_MODIFY
$ icapnotify :== $ICAP_NOTIFY
$ icapstatus :== $ICAP_STAT
```

Command options are specified as described in the HP-UX documentation for each command.

OpenVMS Command Mapping

The following table shows the HP-UX iCAP commands and their OpenVMS equivalents.

Table: HP-UX and OpenVMS Command Equivalents

HP-UX Style	OpenVMS Style
`icapstatus`	`icap show status`
`icapstatus -s`	`icap show status/snapshot`
`icapmodify -C <codeword>`	`icap apply "codeword"`
`icapmodify -c <address>`	`icap set email/contact="address"`
`icapmodify -f <address>`	`icap set email/from="address"`
`icapmodify -i <id>`	`icap set system_id "id"`
`icapmodify -r`	`icap reconcile`
`icapmodify -w <days>`	`icap set warning_days "days"`
`icapmodify -a`	`icap activate /cpu=/defer/ticap`
`icapmodify -d`	`icap deactivate /cpu/defer`
`icapmodify -s`	`icap set active_cpu  n`
`icapnotify -a`	`icap set asset/state=on/off`
`icapnotify -n`	`icap set notification/state=on/off`
`icapmanage -i -u <rule_file>`	`icap manage install <rule_file>`
`icapmanage -C <codeword>`	`icap manage codeword <codeword>`
`icapmanage -a -g <group_name>`	`icap manage add group <group_name>`
`icapmanage -r -g <group_name>`	`icap manage remove group <group_name>`
`icapmanage -T <host>[,<host>]...[-g <group_name>]`	`icap manage test <host,host,...> [/group=<group_name>]`
`icapmanage -a -m <member_name>:<host>[,<host>]... -g <group_name>`	`icap manage add member <member_name> /host_list=(host,host,...) [/group=<group_name>]`
`icapmanage -r -m <member_name>`	`icap manage remove member <member_name>`
`icapmanage -s -g <group_name> [-b] [-v]`	`icap manage status <group_name> [/BRIEF] [/FULL]`
`icapmanage -R [<host>[,<host>]...] [-U <rule_file>]`	`icap manage report <host,host,...> <rule_file>`
`icapmanage -x <host>`	`icap manage extract <host>`

DCL ICAP Commands

The ICAP command supports six command options to perform iCAP operations on OpenVMS systems.

ICAP ACTIVATE

Name
ICAP ACTIVATE - Immediately activates additional cores on the system. (HP-UX equivalent: `icapmodify -a`)

Format
`ICAP ACTIVATE /CPU=n [/DEFER] [/TICAP]`

Qualifiers

`/CPU=n` Specifies the number of additional cores to activate. This qualifier is required.

`[/DEFER]` Defers the activation until the next reboot. (HP-UX equivalent: `-D` option)

`[/TICAP]` Authorize the use of temporary capacity to satisfy this activation request. (HP-UX equivalent: `-t` option)

ICAP APPLY
Name

ICAP APPLY - Apply an iCAP codeword. (HP-UX equivalent: `icapmodify -C`)

Format

`ICAP APPLY "codeword"`

Parameter

`"codeword"` An iCAP codeword obtained from the HP Utility Pricing Solutions portal. Enclose the codeword in double quotation marks.

ICAP DEACTIVATE
Name

ICAP DEACTIVATE - Deactivates cores on the system. (HP-UX equivalent: `icapmodify -d`)

Format

`ICAP DEACTIVATE /CPU=n [qualifiers]`

Qualifiers

`/CPU=n` Specifies the number of cores to deactivate. This qualifier is required.

`/DEFER` Defers the deactivation until the next shutdown. (HP-UX equivalent: `-D` option)

ICAP RECONCILE
Name

ICAP RECONCILE - Activates or deactivates cores (subject to compliance limits) to bring the system to a state where the `intended active` number of cores are active. (HP-UX equivalent: `icapmodify -r`)

Format

`ICAP RECONCILE`

ICAP SET
Name

ICAP SET - Sets various iCAP management variables.

Format

```
ICAP SET parameter [qualifiers]
```

Parameters

ACTIVE_CPU cores.
Sets the number of active cores and the number of intended active cores.

 (HP-UX equivalent:`icapmodify -s`)

Format
```
ICAP SET ACTIVE_CPU count
```

Value
`count`: the number of cores to set active in the npartition.

ASSET
Sets the asset reporting email on or off.
(HP-UX equivalent: `icapnotify -a`)

Format
```
ICAP SET ASSET [qualifier]
```

Qualifiers
`/STATE=state`: specify ON or OFF for the state qualifier value.

EMAIL
Sets the system contact email addresses.
(HP-UX equivalent: `icapmodify -c`)

Format
```
ICAP SET EMAIL qualifiers
```

Qualifiers
`/CONTACT`: The email address that receives the configuration change notifications and exception reports.
`/FROM`: The From address for the email sent from the iCAP system.

NOTIFICATION
Sets the iCAP change configuration email notifications on or off.
(HP-UX equivalent: `icapnotify -n`)

Format
```
ICAP SET NOTIFICATION [qualifier]
```

Qualifiers
`/STATE=state`: specify `ON` or `OFF` for the state qualifier value.

SYSTEM_ID   Sets the system identification used for iCAP asset
reporting.

                          (HP-UX equivalent: `icapmodify -I`)

                          Format
                          `ICAP SET SYSTEM_ID "id"`

                          Value
                          `id`: A user-defined string to identify this system when tracking or
                          reporting usage. Specify a null string ("") to set the system ID to
                          the default value. The default value is the local hostname.

WARNING_DAYS   Sets the temporary capacity warning period to the number of days
specified.

                          (HP-UX equivalent: `icapmodify -w`)

                          Format
                          `ICAP SET WARNING_DAYS days`

                          Value
                          `days`: the number of days of temporary capacity before temporary
                          capacity expiration warning email is sent to the system contact.

ICAP SHOW

Name
ICAP SHOW - Show the status and settings of the iCAP software on the OpenVMS
system.
(HP-UX equivalent: `icapstatus`)

Format
`ICAP SHOW STATUS [qualifiers]`

Parameter
`STATUS` Show the iCAP status and system settings to the standard output device.

Qualifiers
`/SNAPSHOT` Creates a string of snapshot information containing encrypted audit data
and
displays the string to the standard output device. (HP-UX equivalent: `icapstatus -s`
)

ICAP_SERVER

Name
ICAP_SERVER - iCAP server process.

Description

The `ICAP_SERVER` process performs the same functions as the icapd daemon process on HP-UX systems. For more information, see the HP-UX `icapd` manpage. To ensure compliance, the `ICAP_SERVER` is always running on OpenVMS systems in an iCAP complex.

For more information, refer the [HP Instant Capacity User's Guide for Versions 8.x.](#)

Reference Documentation

The following list provides links to some references:

- [HP OpenVMS Version 8.3-1H1 for Integrity Servers Upgrade and Installation Manual](#).
- [HP WBEM products](#)
- [HP Utility Pricing Solutions portal](#)
  [HP Instant Capacity User's Guide for Versions 8.x](#)

A Starlet[1] is Born: New Options for VAX and Alpha Hardware Replacement

Camiel Vanderhoeven, Hardware Illusionist

---

[1] Starlet was the code name for the program that developed the VMS operating system. See the OpenVMS Wikipedia entry http://en.wikipedia.org/wiki/OpenVMS) for additional information.

Introduction

Those who are looking for options to replace aging VAX and Alpha hardware should be aware of the arrival of a new player in the field. Migration Specialties International, a respected OpenVMS consulting firm that is well known for its legacy hardware replacement options, RPG compiler and other migration aids, has teamed up with a number of partners to deliver its own suite of software-based VAX and Alpha hardware emulators.

For this suite of emulators, we've defined an underlying architecture that will allow us to add different emulated systems and options to the suite with an unprecedented degree of flexibility.

This article, written by the lead architect, will focus on the internal architecture designed to support these new emulators.

Goals

We will create a software platform that can virtualize a variety of Alpha and VAX hardware. We want to be able to emulate enough different systems to provide viable alternatives to any existing hardware configuration, including multi-CPU systems.

Our emulators will support OpenVMS (VAX and Alpha emulators) and Digital UNIX/Tru64 UNIX (Alpha emulators only) as operating systems running on top of the virtual hardware.

Our emulators will be hosted on Integrity servers running OpenVMS and Proliant servers running Windows. We will consider supporting a Linux version of the product at a later stage.

When run on OpenVMS/Integrity as the host platform, our VAX and Alpha emulators running OpenVMS will offer the same high-availability features that real VAX and Alpha systems running OpenVMS have to offer.

In the future, we will explore the possibility of coupling our emulators with hardware bus support to enable the use of the emulator with custom hardware interfaces. It would be conceivable to see one of our VAX emulators with an attached Q-Bus or XMI card cage used for replacement of factory automation systems.

We are targeting a production release for a first Alpha emulator in early 2010.

Emulator Design

This section provides a high-level overview of the emulator and shows how the various bits and pieces fit together.
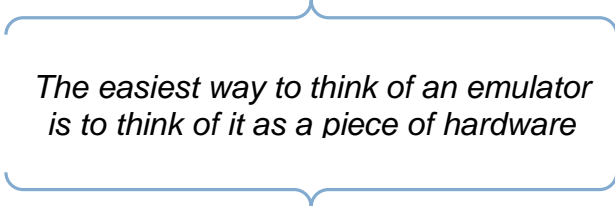
Component Hierarchy

The easiest way to think of an emulator is to think of it as a piece of hardware because that is what it acts like to the operating system and other software running on top of it. Like the real hardware, the emulator consists of modules (components) that interact with each other. Most of the components correspond directly to physical hardware components.
Components have a parent-child relationship to each other. Child components are usually connected to their parent through a virtual bus. For example, disk components are children of a disk controller component, and PCI device components are children of a PCI controller.

Emulator Component

This abstraction poses a problem at the top-level of the emulated system. Most systems have a top-level bus that has no real controller to act as its parent. For example, in the AlphaServer ES40, the top-level bus consists of the D-chips that connect the CPU's to the C-chips (system chipset), the P-Chips (PCI controllers), and main memory. Therefore, the decision was made to create both a VAX emulator component and an Alpha emulator component. These emulator components act as the controller for the top-level bus. For ease of implementation, these components also include main memory.

*The easiest way to think of an emulator
is to think of it as a piece of hardware*

Master Control Program (MCP)

Finally, different emulators and components like networks need to be tied together. This is accomplished through the master control program component. (Are there any fans of either Burroughs B5000 mainframes or the movie *Tron* out there?)

The following (simplified) image shows the components used to emulate an AlphaServer 400 and a MicroVAX, interconnected through an Ethernet network that is also connected to the outside world through one of the host system's network interfaces. The yellow components are those that interact with the outside world. The red line indicates the "extra" parent-child relationship between the network interface cards and the network top-level component.
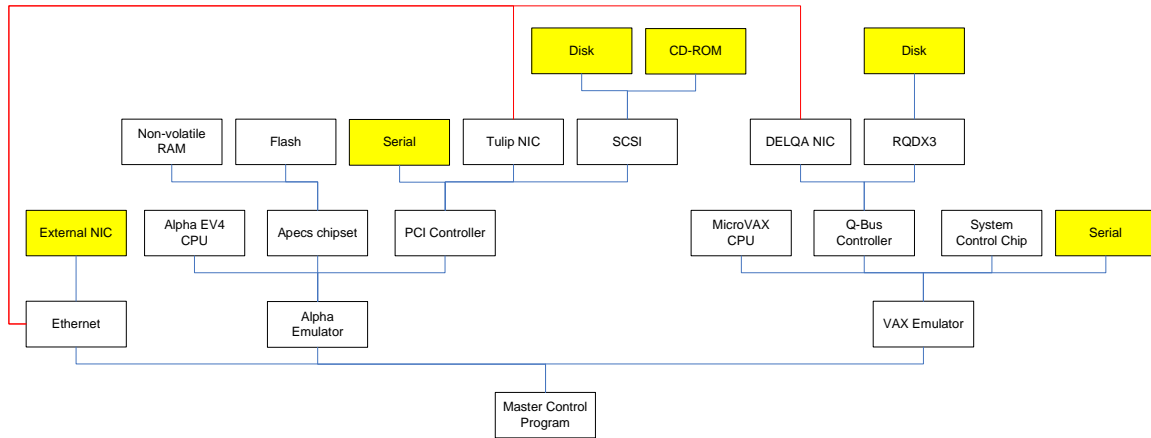
Virtualization Layer

From the beginning, the emulator was written to be very flexible. We first created a framework to be used for writing different emulators. All forthcoming VAX and Alpha emulators will use this common framework. Into this framework, we incorporated all of the functions that all or most emulators will be likely to need, such as:

- Functions for configuring the emulator: instantiation, configuration, and connecting together of all emulator components;
- Functions for controlling the emulator: structured, sequenced discovery, initialization, starting and stopping of all emulator components;
- Emulated Ethernet connectivity between emulators;
- Common interfaces to the outside world for networking, hard-disk emulation and I/O components: for example, communications ports provide the ability to communicate through a telnet session or a physical serial port. This way, this functionality can be shared by any emulated communications port without requiring additional effort, simplifying the emulation environment and providing a more consistent user experience;
- Hiding differences between different host systems from the emulator components, so the same emulator will run on both OpenVMS and Windows;
- Support for making use of multi-CPU or multi-core host systems by threading;
- Emulator licensing and protection.

In short, these are all the emulator functions that are not directly related to the bits, bytes and registers of the emulated hardware. We've named this framework the "Virtualization Layer" because it creates a complete virtual environment for the individual emulators.

Hardware Platform Abstraction Layer

As we want our emulators to run on both Windows on Proliant servers and OpenVMS on Integrity servers, we were confronted with the fact that Windows and VMS behave differently. To avoid having to write platform-specific code for each emulator

component, we implemented an abstraction layer as part of the virtualization layer that hides these differences from the rest of our code. These differences are mainly in the following areas:

- Threading and locking. On VMS, we use the Pthreads library; on Windows, we use the Windows API.
- Physical device access. On VMS, we use QIO's; on Windows, we use various API's.
- Timekeeping.

Putting all platform-dependent code in one place helps us to keep our code base clean.

Use of Object-Oriented Programming (OOP)

The emulator makes extensive use of OOP, particularly of the features offered by the C++ language. While C and C++ are reviled by some for their perceived cryptic nature (although there is no rule that says C or C++ code has to be cryptic), they are commonly considered to be the languages of choice for low-level, portable programming found in operating systems, device drivers, and emulators. C and C++ give programmers a level of control over the bits and bytes of their code few other high-level languages offer, and C and C++ compilers that produce blazingly fast code are available for virtually any platform.

Classes

All components are implemented as classes. That means that a class has been designed for each different kind of emulated component. For instance, if a RQDX3 controller needs to be emulated, a RQDX3 class will be written. Once the class exists, the emulator can create as many instances of that class as required. For example, to emulate a VAX with three RQDX3 controllers, three instances of the RQDX3 class would be generated.

Inheritance

The RQDX3 controller needs to be able to interface with the Q-bus controller and vice versa. The mechanisms involved are the same for all Q-bus devices; the way the RQDX3 communicates with the Q-bus controller is no different than the way a DELQA network interface communicates with it. Because of this shared behaviour, all Q-Bus components share a common base class, the Q-Bus Device base class. This way, the Q-Bus controller can address each of its child components as Q-Bus Devices, rather than as individual types of interface. This takes full advantage of the power of inheritance, a defining feature of OOP.

C and C++ give programmers a level of control over the bits and bytes of their code few other high-level languages have to offer.

Component Base Class

The Q-Bus Device class in turn has the Component class as its base class. The Component class is part of the framework, and provides for such basic emulator-wide functions as naming components, creating parent-child relationships, initializing, stopping and starting the emulator.

Multiple Inheritance

It gets trickier though. Besides being a Q-Bus device, the RQDX3 is also a disk controller. As not all disk controllers are Q-Bus devices (for example, the KZPAA SCSI disk controller is a PCI device), the disk controller base class can't have the Q-Bus device class as its parent. So, the RQDX3 needs to have both the Q-Bus device class and the disk controller class as its base classes. This is called multiple-inheritance.

In the case of the KZPAA SCSI controller, it is even more complicated; it inherits from PCI device, Disk controller, and SCSI device classes. The following diagram illustrates this:
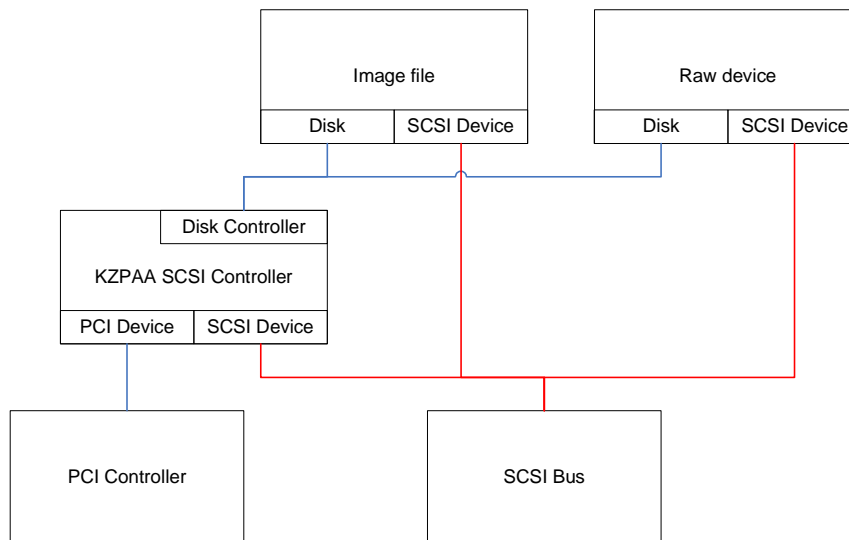
Conclusion

Like computer hardware or operating systems, successful, scalable and adaptable hardware emulation requires an underlying, well-defined architecture. This architecture is

the foundation for the entire product. We have spent considerable effort to define a flexible architecture for our emulators and, hopefully, we've shown you some of its interesting properties in this article.

For More Information

For more information and updates about the upcoming multi-platform VAX and Alpha emulator discussed in this article, visit
http://www.migrationspecialties.com/VAXAlphaEmulator.html.
For more information about the author, visit http://www.camicom.com.
For more information about the open-source ES40 emulator or to download its source code, visit http://www.es40.org.

OpenVMS Technical Journal V13

The MultiNet Intrusion Prevention System

Jeremy Begg, Managing Director
VSM Software Services Pty Ltd.

Introduction

MultiNet® is a TCP/IP network stack for OpenVMS which runs on all OpenVMS hardware platforms. It was initially created by TGV Inc as a VAX/VMS port of the BSD Unix TCP/IP stack, and has for the past 12 years been owned and enhanced by Process Software LLC[6].

The most recent release (V5.3, February 2009) is based on BSD 4.4 and includes an interesting new security feature, the MultiNet Intrusion Prevention System, or "IPS". IPS enables the system manager to configure MultiNet so that it detects intrusion attempts and then blocks further access from the remote system that is the source of the attack. Several of the commonly-targeted MultiNet services such as SSH, FTP and TELNET come equipped to use IPS and mechanisms are provided to allow any network application to do likewise.

This article describes the author's early experiences with IPS in a production environment.

Why Use IPS?

VSM Software Services Pty Ltd runs an AlphaServer DS20E to provide services to customers on the Internet including DNS, webserving and email. We also allow incoming FTP and SSH access for server management and customer website maintenance. In addition to the DS20E we also have a number of VAX, Alpha and Integrity systems for product support & development.

There is no doubt that the Internet has become a hostile place, even for servers running OpenVMS. In operating our services we regularly see VMS breakin events logged by MultiNet's SSH and FTP servers and by PMDF's POP3, IMAP and SMTP servers[7]. We also see suspicious DNS update attempts but (fortunately for us!) not much in the way of Denial-of-Service attacks (*i.e.* attempts to flood the server with so much traffic that it can't maintain acceptable responsiveness).

To date we have used a variety of methods to control this activity, all of them relying on manual observation of an attack and manual updating of a configuration file. For example,

- PORT_ACCESS mapping rules in PMDF to block access from persistently annoying systems
- Configuring DNS to refuse recursive lookups from outside the local network
- Manually updating MULTINET:FILTER-SE0.DAT to block all IP traffic from specified hosts.

---

[6] MultiNet is a registered trademark and Process Software and the Process Software logo are trademarks of Process Software.
[7] PMDF is a standards-based, robust, electronic mail platform and gateway for OpenVMS, Solaris, Tru64 Unix, Windows and Linux. PMDF has been supported and developed by Process Software since October 2000.

MultiNet IPS uses information provided by network services to detect suspicious activity and then dynamically updates the MultiNet packet filter to disrupt that activity. (In this context the term "network service" means a server process on the local system which provides a TCP/IP application service such as SSH, POP3, HTTP, etc.)

We already had a lot of experience with MultiNet's packet filter mechanism so the ability to have it automatically updated by IPS was very appealing.

Configuring IPS

IPS is documented in Chapter 32 of the MultiNet V5.3 Installation & Administration Guide. It is installed as part of the standard MultiNet installation but until its configuration files are set up it doesn't do anything. These files are described in detail in the MultiNet documentation but briefly the procedure is as follows:

1. Copy MULTINET:FILTER_SERVER_CONFIG.TEMPLATE to MULTINET:FILTER_SERVER_CONFIG.TXT.
2. Edit MULTINET:FILTER_SERVER_CONFIG.TXT and set desired configuration options. In particular, use INCLUDE statements to specify the service-specific configuration files which will be loaded when IPS starts.
3. Set up each of the service-specific configuration files specified in step 2. For example, copy MULTINET:SSH_FILTER_CONFIG.TEMPLATE to MULTINET:SSH_FILTER_CONFIG.TXT.
4. Edit each of the service-specific configuration files to specify the criteria for each service, such as the template packet filter rule and the trigger for activating the packet filter.
5. When all configuration files have been prepared, issue the command
   ```
   $ MULTINET SET/IPS/RELOAD
   ```
   to activate them.

Services Protected by IPS

The MultiNet installation kit comes with templates for protecting the most common MultiNet services including FTP, IMAP, POP3, REXEC, RLOGIN, RSHELL, SMTP, SNMP, SSH and TELNET.

OpenVMS systems which are running MultiNet V5.3 and PMDF V6.4 can use IPS to protect the PMDF IMAP, POP3 and SMTP servers. The configuration files for these are shipped in the PMDF_TABLE: directory as part of the standard PMDF V6.4 installation.

The service-specific configuration file

There is a configuration file for each service. Reasonable defaults are provided in the .TEMPLATE files but some changes are required:

`destination_address`   This must match the IP address (in CIDR format) of the interface to be monitored for intrusion activity. For example,

```
        destination_address 150.101.13.12/27
```
This specifies that the interface with IP address 150.101.13.12 is to be monitored. The
subnet portion of the address (/27 in this case) is required but ignored.

`exclude_address`   This specifies one or more remote IP addresses which are
to be ignored, *i.e.* the packet filter will never block those addresses. The template files
come with this configured to block a commonly-used local address (192.168.0.10/24) but
sites may wish to remove or change it.

Each template file also specifies a prototype packet filter entry which looks a little odd at
first glance:

```
        proto_filter "deny tcp 192.168.0.100/24 192.168.0.1/24 log"
```

The two IP address ranges in CIDR format will be automatically replaced by the actual
source and destination addresses when the rule is activated by MultiNet IPS. This rule
only needs to be changed if you wish to change the action, *e.g.* from "deny" to "drop".

Monitoring IPS

The Intrusion Prevention System generates a wealth of evidence for its effectiveness
including assorted MultiNet log files, OPCOM, SNMP and the OpenVMS Security Audit
logs.

There are several log files created by MultiNet IPS:

- MULTINET:FILTER_SERVER.OUT is the primary log file for the filter server
  process.
- MULTINET:FILTER_SERVER_HOURLY_LOG.*yyyymmdd* is a "day file"
  containing a summary of filter actions each hour during the day. A new file is
  created every day at 1am.

Here is an extract from the FILTER_SERVER.OUT file on one of our systems:

```
     FILTER_SERVER V1.0.0

     20-MAR-2009 12:14:07.28 -      Using configuration file
MULTINET_ROOT:[MULTINET]FILTER_SERVER_CONFIG.TXT;
     20-MAR-2009 12:14:07.30 -      Processing include file
"multinet:ssh_filter_config.txt"
     20-MAR-2009 12:14:07.30 -      Using configuration file
MULTINET_ROOT:[MULTINET]SSH_FILTER_CONFIG.TXT;
     20-MAR-2009 20:37:03.46 -      Event message received
     20-MAR-2009 20:37:03.46 -      Component: SSH
     20-MAR-2009 20:37:03.46 -      Rule     : SSH_BOGUS_ID
     20-MAR-2009 20:37:03.46 -      Time     : 20-MAR-2009 20:37:03.46
     20-MAR-2009 20:37:03.47 -      Src Port : 54232
     20-MAR-2009 20:37:03.47 -      Src Addr : 68.54.152.69
     20-MAR-2009 20:37:03.47 -      Dst Addr : 150.101.13.12
```

```
     20-MAR-2009 20:37:03.47 -      Process  : SSHD Master
     20-MAR-2009 20:37:03.47 -      PID      : 208000AD
     20-MAR-2009 21:12:01.31 -      Event message received
     20-MAR-2009 21:12:01.31 -      Component: SSH
     20-MAR-2009 21:12:01.31 -      Rule     : SSH_INVALIDUSER
     20-MAR-2009 21:12:01.31 -      Time     : 20-MAR-2009 21:12:01.31
     20-MAR-2009 21:12:01.32 -      Src Port : 55839
     20-MAR-2009 21:12:01.32 -      Src Addr : 68.54.152.69
     20-MAR-2009 21:12:01.32 -      Dst Addr : 150.101.13.12
     20-MAR-2009 21:12:01.32 -      Process  : SSHD 0000
     20-MAR-2009 21:12:01.32 -      PID      : 20800B83
     20-MAR-2009 21:12:06.49 -      Event message received
     20-MAR-2009 21:12:06.49 -      Component: SSH
     20-MAR-2009 21:12:06.49 -      Rule     : SSH_INVALIDUSER
     20-MAR-2009 21:12:06.49 -      Time     : 20-MAR-2009 21:12:06.49
     20-MAR-2009 21:12:06.50 -      Src Port : 55991
     20-MAR-2009 21:12:06.50 -      Src Addr : 68.54.152.69
     20-MAR-2009 21:12:06.50 -      Dst Addr : 150.101.13.12
     20-MAR-2009 21:12:06.50 -      Process  : SSHD 0001
     20-MAR-2009 21:12:06.50 -      PID      : 20800C04
.
... Event messages removed for brevity ...
.
     20-MAR-2009 21:12:47.13 -      Event message received
     20-MAR-2009 21:12:47.13 -      Component: SSH
     20-MAR-2009 21:12:47.13 -      Rule     : SSH_INVALIDUSER
     20-MAR-2009 21:12:47.13 -      Time     : 20-MAR-2009 21:12:47.12
     20-MAR-2009 21:12:47.13 -      Src Port : 50039
     20-MAR-2009 21:12:47.13 -      Src Addr : 68.54.152.69
     20-MAR-2009 21:12:47.14 -      Dst Addr : 150.101.13.12
     20-MAR-2009 21:12:47.14 -      Process  : SSHD 0009
     20-MAR-2009 21:12:47.14 -      PID      : 20800C20
     20-MAR-2009 21:12:47.14 -      Creating a filter for component ssh
rule ssh_invaliduser
     20-MAR-2009 21:12:47.14 -      src address   =  68.54.152.69/32
     20-MAR-2009 21:12:47.14 -      dst address   =  150.101.13.12/27
     20-MAR-2009 21:12:47.14 -      interface     =  se0
     20-MAR-2009 21:12:47.14 -      filter expires   20-MAR-2009
21:17:47.14
     21-MAR-2009 00:00:00.50 -      Performing daily maintenance
```

The extract above shows that the filter server started at 20-MAR-2009 12:14:07.28 and
loaded a single service-specific configuration file (for SSH). At 20:37:03.46 the SSH
server reported suspicious activity but this was not followed by any other such activity
within the specified timeout (5 minutes by default) and so was ignored. Then at 21:12:01
the SSH server reported more suspicious activity and this time the remote system
(68.54.152.69) persisted in its breakin attempts. After ten such reports in the space of
under a minute the IPS created a packet filter to block the remote system.

The FILTER_SERVER_HOURLY_LOG files contain hourly snapshots of IPS activity.
For example the time period corresponding to the extract above looks like this:

```
Filter server hourly snapshot for hour 21 of 03/20/2009

Component  ssh
```

```
    Rule ssh_bogus_id
        number of hits:        0
        destination address: 150.101.13.12/27

        Address 68.54.152.69/32
            number of still-queued events:   0
            number of all events:            0
            number of filters created:       0
            Address entry to be deleted:     21-MAR-2009 00:42:03.46

    Rule ssh_authfailed
        number of hits:        0
        destination address: 150.101.13.12/27

    Rule ssh_userauth
        number of hits:        0
        destination address: 150.101.13.12/27

    Rule ssh_invaliduser
        number of hits:       10
        destination address: 150.101.13.12/27

        Address 68.54.152.69/32
            number of still-queued events:   0
            number of all events:           10
            number of filters created:       1
            Address entry to be deleted:     21-MAR-2009 01:12:47.19
```

In addition to the log files, the regular MultiNet interface commands can be used to see what packet filters are in place at any given moment:

```
$ mu show/int se0/filter
Device se0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST,D2>
        VMS Device = EWA0
        IP Address = 150.101.13.12
        No common links defined

MultiNet Packet Filter List for se0:

Logging is disabled


                       Source Address / Port
Action  Proto   Hits  Destination Address / Port
------  -----   ----- -------------------------------------------------
--
drop     tcp     29   213.174.151.17/32
                      150.101.13.0/27
                      FLTSVR,LOG
$
```

(Note that the output above was generated some days after the events shown in the log file extracts.)

Closing Remarks

We're still in the early days of using IPS here at VSM but it's already proven to be effective at limiting intrusion activity. In general the default settings are very good but some adjustments might give better results, and here are a few suggestions.

- In setting up the SSH filters for IPS we chose to change the default *proto_filter* rule from "deny" to "drop". In our experience with SSH-based attacks the remote systems keep sending packets to the server even though they're getting "administratively denied" responses. Changing the rule to "drop" causes nothing whatsoever to be sent back to the remote systems, and they seem to stop trying much sooner.

- The default trigger for many events is 10 occurrences inside 5 minutes. At our site where most SSH users are what you might call "sophisticated" we could probably tighten this to (say) three login failures inside 5 minutes. On the other hand if we had a large number of users the probability that any given user would enter the wrong password would be somewhat higher, and "3 in 5" might be too restrictive.

- Once we get PMDF upgraded on our primary server we will look at implementing IPS for PMDF's IMAP, POP3 and SMTP servers. All of them are popular targets for password-hunting netbots.

If you run MultiNet at your site and you suffer from intrusion attempts, have a look at IPS - it might be just what you need!

For More Information

The author will be happy to field enquiries of a technical nature about MultiNet IPS or indeed any aspect of MultiNet or PMDF; please send your enquiry to
jeremy@vsm.com.au (For technical support about a specific problem please use your usual support channel.)
For general enquiries about MultiNet, PMDF or any other Process Software product please visit the Process Software web site, http://www.process.com/.

The MultiNet documentation can be viewed on-line at
http://www.process.com/tcpip/mndocs.html
and the IPS documentation can be found at
http://www.process.com/tcpip/mndocs53/ADMIN_GUIDE/ch32.htm

OpenVMS and Perl – a Powerful Match

Bernd Ulmann

Introduction – OpenVMS and Perl a Powerful Match

Although OpenVMS has a powerful command language with its DCL interpreter, there are everyday tasks which can be solved way more easily using another interpreted language like Perl which is readily available for OpenVMS on all three architectures. The following article is the result of my experiences with Perl on this platform as well as some talks covering this topic I delivered at DECUS and HP in 2008 and 2009. All examples in the following were chosen from my own daily work on a large OpenVMS system and range from simple code snippets, programs for one time usage to larger Perl programs running as batch jobs and performing crucial tasks like fetching mail from a POP3-server and the like.

To make one point clear at the beginning: Perl is not a replacement for DCL but it makes life much more easier when it comes to parsing and modifying files, socket communications, data base access etc.

Some Basics about Perl

Let us start with summing up some facts about Perl – although Perl is way too powerful to be described exhaustingly in a few pages like this, at least an impression of some of its basic features may be given for those who have never seen or written any actual Perl code.

First of all there is one main source of information and enlightenment for the Perl programmer, the so called Camel Book, "Programming Perl", published by O'Reilly. This book is a joy to read and contains a wealth of Perl know how with lots of practical examples and should not be missing on every desk of a Perl programmer, even an experienced one.

But now to the promised basic facts about Perl: First of all, Perl was developed initially by Larry Wall and its name is not an acronym but a retronym with a variety of interpretations (some people claim that it stands for "Pathological Eclectic Rubbish Lister" but "Practical Extraction and Report Language" fits better). Perl is a modern interpreter language – in fact, the interpreter does a terrific job so Perl programs tend to run surprisingly fast, even on slow machines like a VAX (compare that with Java which does not run on a VAX at all). Furthermore Perl runs on a wide variety of architectures and operating systems (maybe even more than Java is available for) making it a language for really portable applications. Perl is extremely powerful and concise and there is a plethora of modules readily available for nearly all purposes. Perl's basic philosophy is "There is more than one way to do it" (compared with other languages like Python which try to force the programmer to only one way to solve a problem which is good for the bondage and discipline fraction but not for those of us who like to write elegant code).

Most programmers who are not Perl-aware tend to call Perl an unreadable language, which is only true at first sight at best – at a first glimpse Perl code can really look like line noise, but for the initiated Perl programmer code like this is very readable (compare it with APL, eg.).

First of all, Perl does not care about the type of its variables – in fact it is basically a type free language. Instead, Perl cares about the structure of variables – in essence there are three such basic structures:

- Scalars: A scalar variable can hold a single value at a time. The name of such a variable is always prefixed by a dollar sign like in "`$pi = 3.14159265;`".
- Arrays: An array is an indexed list consisting of scalars as its elements. The name of an array variable is prefixed by `@` like in "`my @entries;`".
- Hashes: These are similar to arrays since their elements are scalars, but instead of numerical indices strings are used to address elements within a hash. The name of a hash variable is preceded by `%` as in "`my %data;`" for example.

Some examples for these basic structures are shown in the following:

```perl
my $pi = 'three_point_one_four';
my @array;
$array[0] = 'Something';
$array[1] = 'Something else';
$array[2] = 2.718;
my %data;
$data{'name'} = 'Bernd';
$data{'occupation'} = 'VMS enthusiast';
```

Perl supports all of the common control structures like "`if...else`", "`while`", "`do...while`", "`for`" etc. Every such statement controls a block surrounded by braces which can not be left out like in C if only a single statement is to be controlled, so a typical if-statement has this form:

```perl
if (some_condition)
{
  do_something;
  ...
}
```

If only a single statement is to be controlled, a second form of control can be used, the so called *statement modifier*:

```perl
do_something if condition;
```

This is quite similar to spoken English and helps to keep program sources short. In the following some simple examples of loops are shown:

```perl
for (my $i = 0; $i < 10; $i++) # This is not very common in Perl
{
  print "$i\n";
}
for my $i (0..9) # This looks more like Perl
{
  print "$i\n";
}
print "$_\n" for (0..9); # A for loop as a statement modifier
```

As powerful as these control structures provided by Perl are, much of its power results from its embedded regular expression parser. Regular expressions really tend to look a bit like line noise so no reasonable example will be given now as some may be found in the examples section.

As already mentioned there is a wealth of modules which can be used with Perl readily and which are available at a central location called CPAN, short for "Comprehensive Perl Archive Network", which can be accessed at http://www.cpan.org and http://search.cpan.org. Regardless what your initial problem is, you should always have a look at CPAN first – normally there already is a module which solves at least part of your problem if not the whole problem at all. Examples for the power of Perl's modules are countless as you will see in the examples section.

It is time to give some advices about what to do in general and what not:

Do:
- Be open for the Perl way of solving problems. Perl programs tend to be very short and powerful, especially when compared with equivalent solutions in other programming languages.
- Use functions like `split`, `join`, `map`, `grep` and the like instead of unnecessary C-style loops.
- Use hashes when you need to lookup values.
- Always use regular expressions to parse, manipulate or split (complicated) strings.
- Be strict and use warnings all the time.
- Get the *Camel* Book.

Do not – do not even think about it:
- ...program in Perl like you program in C or Java or DCL or anything else.
- ...use arrays when you can use hashes – especially never ever loop over an array to find an element.
- ...write linear narrative code.

How does one get a running Perl installation on an OpenVMS system (or any other system as well)? Basically there are two ways to get a Perl interpreter up and running on a given system:

1. You can use a precompiled package – for OpenVMS there is a HP-supplied distribution kit, for other platforms there are equivalent such kits available.
2. Get the sources and compile, link and install the system yourself.

Personally, I always prefer the second method since I like to know what is really running on my system and sometimes I want to do things differently compared with a precompiled installation kit. (Compiling a Perl system on an Alpha or Itanium system is fast, but on a VAX this can take several hours, so be prepared!)

As all of you know (and love), OpenVMS is different from (and superior to :-)) other operating systems which has to be taken into account when porting or writing software. There are quite some modules to be found on CPAN which encapsulate OpenVMS specific tasks like interfacing the mail system etc. In addition to that there are modules to handle operating system specific tasks as transparently as possible which should be used whenever possible to yield operating system agnostic code.

In the following a short and definitely incomplete list of modules is shown which are especially useful in an OpenVMS environment:

- `VMS::Device` – interface to `$GETDVI` and the like.
- `VMS::Filespec` – converts between OpenVMS and UNIX filespecs.
- `VMS::FlatFile` – use hashes to work with index files.
- `VMS::ICC` – intra cluster communication services.
- `VMS::Mail` – interface to the OpenVMS mail system.
- `VMS::Process` – manage OpenVMS processes.
- `VMS::Queue` – work with queues and their entries.
- `VMS::Stdio` – file operations like binmode, flush, vmsopen etc.
- `VMS::System` – retrieve system information.
- `File::Basename` – system independent operations on filenames and the like.

Having all this said, it is time to show some real live examples of Perl in an OpenVMS environment.

Examples

Some of the following examples, especially the simpler ones, are accompanied with their source code which may be of interest, although more complex examples are only described textual with some code snippets. (If you are interested in the source code of one of these more complex examples, please feel free to contact the author directly by mail.)

Programs for one-time-usage

Many everyday tasks require that system administrators as well as programmers solve unexpected problems like clever pattern matching, parsing logfiles etc. which are not readily handled with standard DCL tools. Many of these problems can be solved on the fly using a couple of lines of Perl code.

Perl can be used as a command line tool like `awk` for example. This can be very useful when you have a puzzling problem which does not deserve a real program but nevertheless needs a clever data conversion on the fly or something like that. Since there are a variety of command line options for Perl which are useful in this context, only simple examples are given in the following (more information may be found elsewhere like the Camel Book).

Adapting configuration files to VMS

Once I inherited a configuration file transfer.ini which looked like this (but contained literally hundreds of such sections):

```
[logging]
  log    = log/transfer.log
  ticket = log/ticket.log
[templates]
  ticket = templates/ticket.tpl
  mail   = templates/mail.tpl
```

Of course, these pathnames are not very OpenVMS like and it would have been quite cumbersome to edit all of the manually. Now one could write a Perl program reading the file, performing the necessary changes using regular expressions and then writing the result back to disk. Since tasks like these are commonplace, Perl can be used as a mighty command line tool for performing in-place edit operations like transforming the pathnames in the example above into valid OpenVMS file names. In the example shown, this was accomplished with the following command line:

```
$ perl -i -pe "s/^(.*\s*)=(\s*)(.+)\/(.+)/$1=$2\[\.$3\]$4/" transfer.ini
```

It looks a bit like line noise, right? What does it do? First of all it loops over all lines in transfer.ini and matches lines which contain an equal sign with a string to its left and two strings separated by a slash to its right. These three strings are captured using parentheses and then the line which matched this expression will be substituted by a new line constructed out of the parts just captured. Applying this single line statement to the configuration file shown above yields a new version of this file with the following structure:

```
[logging]
  log    = [.log]transfer.log
  ticket = [.log]ticket.log
[templated]
  ticket = [.templates]ticket.tpl
  mail   = [.templates]mail.tpl
```

which is exactly what was desired.

Repairing HTML files

Another problem I was faced with was that a user of a WASD web server insisted on creating her web pages using "modern" tools running on a MAC. Unfortunately these particular tools just refused to generate proper HTML encoding for special German characters like "ä" which should be coded as "ä" in HTML. Instead these tools just insert "ä" which results in a completely bogus display of the resulting web page. This problem can be corrected on the fly with a Perl call like this:

```
$ perl -i -pi "s/\xC3\xA4/\&auml\;/g; s/\xC3\xB6/\&ouml\;/g;
s/\xC3\xBC/\&uuml\;/g; s/\xC3\x84/\&Auml\;/g; s/\xC3\x96/\&Ouml\;/
g; s/\xC3\x9C/\&Uuml\;/g; s\xC3\x9F/\&szlig\;/g;" [...]*.html
```

OK – it really looks like line noise, but it can be easily put into a DCL routine which can be called after each upload of HTML files by this particular user and thus correcting the problem on the fly. This very special user (my beloved wife, to be exact) also wanted to include a background picture into her web pages which the tool used just does not support. Using Perl it was simple to extend the generated HTML-code by a proper `background-image`-directive, too.

Making sure that a large LaTeX document is consistent

Another problem came up when I wrote a really large book using LaTeX without using BibTeX (which is stupid, but the project grew from a pet project to a major project and when I realized that the simple bibliography of basic LaTeX was not really powerful enough to cope with the bibliography, it was literally too late to switch to BibTeX). Having a very long list of literature, I feared that some entries could have been rendered unused in the text body due to changes in its structure etc. Although LaTeX tells you when you cite something which is not defined, it does not tell you if you have bibliography entries which are not cited which is annoying.

A typical entry has the form:

```
\bibitem{zachary} %book
```

```
G. Pascal Zachary, \emph{Endless Frontier – Vannevar Bush,
Engineer of the American Century},
```

```
The MIT Press, 1999
```

while a citation looks like

```
cf. \cite{zachary}[p.~142]
```

Having a document with more than 120000 lines of LaTeX code resulting in about 600 pages of text with more than 600 bibliography entries, a solution was necessary to make sure that no entry went uncited. This was accomplished with the following Perl program which reads in the complete LaTeX source code with a single statement and parses this for all citations in a first run while building a hash containing these citations. In a second run through this data all bibliography entries are processed and a message is printed for every bibliography entry without a corresponding citation:

```
use strict;
use warnings;

die "Usage bib.pl <filename.tex>\n" unless @ARGV + 0;

my $data;
open my $fh, '<', $ARGV[0] or die "Could not open $ARGV[0]: $!\n";
{
  local $/;
  $data = <$fh>;
}
close $fh;

my %cite;
$cite{$_}++ for $data =~ m/\\cite\{(.+?)\}/g;

$cite{$_} or print "$_\n" for $data =~ m/\\bibitem\{(.+?)\}/g;
```

Parsing a Log File and Generating Some Statistics

Some months ago I had to parse a log file containing entries like these:

```
[LOG|SYSTEM|2008 May 13, 14:15:26 (886)|ENGINE.batch]
Loaded 16 events in 497 milliSecs
[END]
[LOG|SYSTEM|2008 May 13, 14:15:55 (281)|Risk|BatchJob]
Time to execute Scenario 24902 ms
[END]
[LOG|SYSTEM|2008 May 13, 14:15.55 (283)|Risk|BatchJobThread]
Time to execute Scenario 13662 ms
[END]
```

I was asked to calculate the arithmetic mean and possibly other values of the time necessary to execute scenarios and thus wrote the following short Perl program:

```
use strict;
use warnings;
die "Usage: stat3 \"yyyy mm dd\" \"hh:mm:ss\"\n" if @ARGV != 3;

my ($file, $date, $min_time) = @ARGV;
my @values;
open my $fh, '<', $file or die "Could not open $file: $!\n";
```

```
{
  local $/ = '[END]';
  while (my $entry = <$fh>)
  {
    my ($time, $duration) = $entry =~
      m/^.+\|.+\|$date,\s(\d\d:\d\d:\d\d\s).*execute
Scenario\s(\d+)\sms/s;
    push (@values, $duration) if $time and $time ge $min_time;
  }
}
close $fh;


print 'Average: ', in(eval(join('+'. @values)) / @values) / 1e3,
  ' s (', @values + 0, ")\n" if @values + 0;
```

Of course, the simple arithmetic mean could have been calculated in the main loop
without the need of storing all values captured from the log file into an array but since it
was not clear if more complex calculations might be necessary, it was decided to save all
values first and use them for the calculations in the next step. This led to the funny way
of computing the arithmetic mean by concatenating all array elements in a single long
character string, joined together with '+'-characters. This string is then fed into an eval
which is not efficient but shows what can be done using a dynamic programming
language like Perl.

Are there any files with W:WD-rights on my system disk?

Another day I was asked "How can you be sure there are no files on your system disk
which are writable by WORLD?" Good question – this calls for a short Perl program,
too, which shows how external commands and functions can be called using backticks
while capturing their output into program variables:

```
use strict;
use warnings;


my ($fc, $mc) = (0, 0);
for my $line (`dir/prot/width=(file=60) [...[`)
{
  my ($file, $w) = $line =~ m/(.+)\s+.+,(.*)\)/;
  next unless $file;
  $fc++;
  print "$file\n" and $mc++ if ($w =~ m/[WD]/);
}
```

```
print "$fc files processed, $mc are world writable/deletable!\n";
```

It turned out that no files were endangered by wrong protection settings and yes, the program was tested by creating a file with W:WD rights deliberately.

Migrating a MySQL Database to Oracle/RDB

Another one time script which proved itself being very useful was written to migrate a MySQL database running on a LINUX machine to an Oracle/RDB system running on an OpenVMS system (cf. "Bringing Vegan Recipes to the Web with OpenVMS", OpenVMS Technical Journal, No. 8, June 2006). All out of the box attempts to solve this problem did not work directly due to the very different output/input formats regarding the generated files. A first attempt to transform a MySQL output file into a suitable load file for Oracle/RDB proved to be quite complicated and having much overhead, so it was decided to give up this approach and try an online approach using a simple Perl program to connect to both databases at once, reading data from MySQL and writing it directly into Oracle/RDB.

The resulting program turned out to be quite generic and only expects the necessary database connection parameters as well as a list of tables to be copied. The copy operation itself was faster than expected and even outperformed the first attempt using file based export/import with an external transformation routine implemented in Perl.

Larger Perl Programs

Many problems which occur on a regular basis can be solved using Perl, too. Examples for such problems are:

- Generating simple web server statistics on a daily basis.

- Fetching stock market data from a web server and storing it into a MySQL database.

- Fetching mail from a POP3 server in regular time intervals and distributing these mails to the OpenVMS mail system.

- Sending outgoing mails to an SMTP server requiring authentication which is not currently supported by OpenVMS's TCPIP stack.

- Caching results from database queries to speed up execution time of programs requesting data from a database etc.

These four examples will be described briefly in the following showing the power of Perl in larger applications:

Simple Web Server Statistics

After observing that the WASD web server running on an OpenVMS system was unexpectedly busy, a simple web server statistics was to be programmed to see which files were requested how often. All in all a result like this should be generated:

```
2734: my_machines/dornier/do80/chapter_1.pdf
 288: my_machines/bbc/tisch_analogrechner/anleitung.pdf
 117: publications/anhyb.pdf
  97: publications/handson.pdf
```

This was accomplished after only a couple of minutes with the following short Perl program:

```perl
use strict;
use warnings;

die "File name and account name expected!\n" unless @ARGV == 2;
my ($log_file, $account) = @ARGV;

open my $fh, '<', $log_file or die "Unable to open log file $log_file,
$!\n";

my %matches;
while (my $line = <$fh>)
{
  my ($ip, $key) = $line =~ m/^(\d+\.\d+\.\d+\.\d+).*"GET
\/$account\/(.+?)\s/;
  next if !$ip or $ip =~ '^192.168.31';
  $key =~ s/"//g;
  $key .= 'index.html' if $key =~ m:/$:;
  $matches{$key}++ if $key =~ m/(html|pdf|txt)$/;
}

close $fh;

printf "%5d: %s\n", $matches{$_}, $_
  for (sort {$matches{$b} <=> $matches{$a}} keys(%matches));
```

A couple of months later my friend Michael Monscheuer wrote an equivalent web server statistics script in pure DCL which was much (very much, in fact) longer than the program shown above, although I have to admit that his solution seemed more easy to read at first sight, but due to the sheer amount of code this impression faded rather quickly.

Fetching Mail from a POP3-server

Sometimes it would be desirable to fetch mails from a standard POP3-server and make these mails available in the OpenVMS mail system so the system's users can access their mails using MAIL or a suitable webinterface like yahmail or soymail. To make this possible, a Perl written batch job is required which polls in regular intervals a variety of POP3-servers and their associated mailboxes, fetches mails and distributes these mails to the various users of the OpenVMS system.

The overall Perl code for implementing this batch job consists of only 140 lines since most of the really complicated subtasks were already implemented in the following modules readily found on CPAN:

- `Net::POP3` – client interface to the POP3-protocol.
- `IO::File` – file creation and access methods.
- `POSIX qw(tmpnam)` – used to create temporary file names.
- `VMS::Mail` – interface to the OpenVMS mail system.

When it is possible to receive mails, it would be nice to be able to send mails, too, as the following example shows:

SMTP-Proxy

Almost every current mail provider requires that its clients authenticate prior to sending mail via their SMTP server(s). Unfortunately, authentication is not supported by the TCPIP package for OpenVMS. Since a requirement was to send output mail directly from the OpenVMS system, i.e. without an intermediate proxy system like a LINUX host or the like, it was decided to implement a small SMTP-proxy in Perl running on the OpenVMS system itself.

This proxy connects on the local machine to port 25 and listens for outgoing mail while another connection is maintained to port 25 of the provider. Every outgoing mail is parsed and enriched with the necessary authentication information before being sent to the provider which solved the initial problem quite easily.

This SMTP-proxy makes use of the following modules yielding an overall code size of only 68 lines of Perl code:

- `Net::ProxyMod` – this module allows easy TCPIP-packet modification.
- `MIME::Base64` – MIME-encoding and -decoding.
- `Tie::RefHash` – allows using references as hash keys.

Database-Proxy

Sometimes it is desirable to perform database accesses not directly but via a proxy which might either contain some business logic and/or cashing mechanisms to reduce database load and to speed up the application at the cost of some additional memory consumption. Since a former article already described this Perl based proxy in detail (cf. "Bringing

Vegan Recipes to the Web with OpenVMS", OpenVMS Technical Journal No. 8, June 2006) only the obtained speedup of a factor of 10 obtained with this proxy should be noted here.

Conclusion

Over the years it turned out that Perl is an invaluable tool for solving everyday problems as well as for writing large and complex programs running interactively as well as in batch mode. Especially in an OpenVMS environment which often poses very special needs when it comes to system connectivity, interfacing and the like, Perl can be employed with much benefit.

Perl does not consume too many resources and is really fast for an interpreted language so it even runs very well on smaller VAX systems (where not even a JVM is available).

It is important to realize that Perl is not just a "scripting language" as it is sometimes called. Instead it is mighty programming language and programming environment, thus Perl should be taken seriously. So, have fun with Perl and OpenVMS – a perfect team for all of us.

For More Information

The author can be reached at ulmann@vaxman.de.

OpenVMS Technical Journal V13

Performance Management for OpenVMS Systems

Jeff Maffe, SightLine Systems Corporation

Introduction

HP's OpenVMS platform is typically used to host mission critical applications with stringent service level agreements – down time is definitely not an option. The personnel responsible for managing the OpenVMS platforms must have accurate, real time metrics and analytic capabilities at their disposal in order to assure that these mission critical systems are always performing the way they were intended to, and that the applications and services hosted on these machines are always available to the user community. The reality is that IT organizations cannot expect to manage OpenVMS and other business critical systems with the commodity solutions available from a myriad of software vendors, these systems need to be managed using software that is purpose built for OpenVMS.

IT organizations face a litany of challenges when looking to effectively manage and monitor any system for performance and capacity planning, let alone the business critical systems responsible for most of the workload and much of the revenue coming into the business, including:

- Understanding the IT resources needed to effectively deliver services to internal and external customers.
- Balancing service requirements and available resources to support those requirements effectively.
- Centralizing data and administrative/management capabilities.
- Rapidly identifying the root cause of performance, bottleneck or response time issues, eliminating finger pointing among departments or teams (IT versus Network, and so on.)
- Preventing outages or performance issues prior to them having a negative impact on business
- Minimizing the time the analysts spend in poring over massive amounts of disparate data from multiple sources.
- Eliminating silos or point solutions without sacrificing management capabilities.
- Extending the life of existing IT assets to reduce costs.
- Getting things accomplished in a proactive, real time fashion as opposed to being in constant reactive mode.

Most performance management and capacity planning software packages do not provide the breadth or the granularity to help the IT organization even come close to addressing these challenges. To make things worse, most IT organizations are using multiple applications to manage the performance of their infrastructure, creating problems around disparate and conflicting data and multiple data sources, burdening even further the analysts that spend most of their day poring over this data.

SightLine for OpenVMS

SightLine Systems has been providing performance and capacity management solutions to large enterprises for almost 30 years, and has been delivering VAX and OpenVMS solutions for 20 years. SightLine's applications are purpose built to support complex, mission critical systems. With SightLine, system administrators can collect current and historical information through a centralized console, providing real time status of your entire OpenVMS environment. SightLine's OpenVMS solution provides several key benefits, including the abilities to:

- Detect, diagnose, prevent and predict data loss and downtime through a wide range of analytics and automation.
- Monitor multiple OpenVMS systems (as well as other mission critical platforms) concurrently.
- Improve overall performance through proactive system tuning and troubleshooting and the ability to track and monitor a full array of OpenVMS specific parameters.
- Provide full support for the performance, reliability and availability offered by OpenVMS and its related resources via monitoring and managing OpenVMS clusters.

Commodity solutions will not allow users to have the visibility needed to manage OpenVMS systems effectively. Because they have been purpose built for OpenVMS, SightLine's solutions allow for the most efficient monitoring of those platforms in real time, and provides the granular, in-depth information needed to manage these systems effectively, including the following:

CPU Statistics

Modes Metrics - The Modes metrics describe CPU utilization both in total and by component parts. Each metric is expressed on a scale that has zero as its minimum and one hundred times the number of active processors (Active CPU Cnt) as its maximum.

States Metrics - The States metrics describe the scheduling State Queues. OpenVMS assigns processes to those queues so that their scheduler can prioritize their use of system resources. SightLine delivers the number of processes in each of these states, as well as total processes on the system. SightLine also provides the count of processes on the COM queue to indicate how many are Batch, Interactive and Network processes.

Memory Statistics

MPW Metrics - The Modified Page Writer (MPW) metrics describe the nature of activity, and performance of the Modified Page Writer Mechanism, which is the portion of the OpenVMS Swapper that maintains the Modified List.

Page Metrics - The Page metrics describe the behavior and performance of the OpenVMS memory management software. The metrics Modified List Size, Free List and Zeroed List Size measure the three respective components of the Secondary Page Cache.

The remainder of the Page metrics reported by SightLine measure the rate at which various memory management activities occur.

Pool Metrics - The Pool metrics describe memory that OpenVMS allocates for its own use and the use of its requesting processes in the pool (including both Paged and Non-Page Pool). Pool metrics include request, expansion, and failure rates for both types of pool.

I/O Statistics

Disk Metrics - Disk metrics describe the level of activity and performance of your disks. All disk metrics except Disk Count are subscripted, which means that for each metric SightLine provides a value for each device on the system. This allows you to display and analyze the performance of each individual disk relative to Operation Count, Queue Length, Disk Errors, Response Time, Disk Space, and Read and Write Rates.

Disk Controller Metrics - SightLine can report on many disk, HotFile and XFC metrics on a per-controller basis, for use by those who want to balance loads between multiple Fibre Channel paths.

FCP Metrics - The File Control Primitive (FCP) metrics describe the performance of the OpenVMS file system. They can be monitored to determine the nature, efficiency, and system impact of file operations.

XQP Metrics – The XQP metrics include call rates, XQP disk read and XQP write rates, Cache hits, CPU time, Window hits, split transfers, XQP page faults, allocations, file creations, volume lock waits, erases, and window turns.

I/O Metrics - The I/O metrics describe system-wide input/output activity. Once you become familiar with their behaviors during periods of normal activity, you can detect abnormalities by setting thresholds on those that affect (or reflect) your system's performance. Using these abnormalities as an investigative starting point, you can quickly pinpoint performance problems within your I/O subsystem. I/O metrics include Direct and Buffered I/O Rate, Log Xlate Rate, File Open Rate, Process Inswaps Rate, and Open File Count.

MSCP Metrics - The MSCP metrics describe the nature of activity, level of activity, and performance of the Mass Storage Control Protocol, which provides cluster-wide access to local devices.

Files Metrics - The Files metrics describe the levels of activity and the performance of the system-wide file system, including the file system caches and other indicators. File system cache metrics include Tries, Hits, Hit Rate, Misses and Index for FIDs, Extent Cache, File Headers, Directory FCBs, Quota, Bitmap, and Directory Data.

Virtual I/O Cache Metrics – The Virtual I/O Cache metrics describe the nature of activity, level of activity, and performance of the Virtual I/O Cache, which was introduced to OpenVMS beginning with v6.0 on VAX and v1.0 on AXP systems. The Virtual I/O Cache is a single, file-oriented cache designed to improve I/O performance on stand-alone and clustered systems. For the Virtual I/O Cache, SightLine can report total pages, bytes, free pages, free bytes, pages in use, bytes in use, read attempts, read hits,

read hit percentages, write attempts, write hits, write hit percentages, read bypasses, write bypasses and files retained.

eXtended File Cache (XFC) Metrics – SightLine collects XFC metrics from OpenVMS v7.3 and later. SightLine gathers all XFC performance data, including XFC Cache, Disk, and I/O Size information.

Network Statistics

DECnet Metrics - The DECnet metrics describe the level of DECnet activity on your local system. DECnet metrics include Arriving Local and Transit Packet Rates, Departing Local Packet Rate, Transit Congestion Loss Rate, and Receive Buffer Failure Rate.

Ethernet Metrics - SightLine provides information about Ethernet Performance and Levels of on the system, including Blocks Sent and Received, Data Overruns, Fails, Errors, and Buffer Availability.

SCS Metrics - The SCS metrics describe the level and activity of performance of the System Communication Services. The SCS metrics are subscripted, which means that for each metric, SightLine measures the activity for each virtual circuit. SCS metrics include Datagram Send and Receive Rates, Message Send and Receive Rates, Connection Queue Rates, and additional metrics related to Block Data Transfers.

SYSGEN Statistics

Dynamic SYSGEN Metrics - The Dynamic SYSGEN category contains System Generation parameters that you can change while the system is running. In other words, you can implement changes to Dynamic SYSGEN parameters without rebooting your system. In some cases, changes to these parameters take effect almost immediately. Other times, changes take effect only after certain non-routine external events occur. Wherever possible, the SightLine data dictionary displays the effective time of any changes you make.

Static SYSGEN Metrics - The Static SYSGEN category contains those System Generation parameters that can be implemented only by changing their values using the OpenVMS SYSGEN (or SYSMAN), or AUTOGEN utility and rebooting the system.

HotFiles Statistics

The HotFiles statistics show the files that have the most activity. The activity may be based on the number of reads/writes or the amount of data read/written. A user-defined minimum "score" can be used to determine the level of activity a file must have for a given interval before it is considered a "HotFile". The user may also choose (by filename) which files to include or exclude from HotFiles report, for example, one could choose to ignore all .EXE files from the statistics.

Workload Statistics

SightLine can be configured so that process data (CPU usage, page faults, Direct I/O, Buffered I/O, average memory usage, image activations and total process count) can be reported based on Groupings you define, dividing the processes into separate workloads

according to UIC, Account Username, Processname Image, Mode or combinations thereof. This makes it much easier to identify rogue users or applications, and is used by some customers for chargeback purposes.

Lock Manager Statistics

SightLine can report rates for new and conversion enqueues, enqueue waits, enqueues not queued, dequeues, blocking ASTs, deadlock searches, deadlock finds, total locks and total resources.

Distributed Lock Manager Statistics

SightLine collects metrics that describe the activity required for the Distributed Lock Manager to synchronize operations across a clustered system. Reported metrics include rates for incoming and outgoing messages in support of each of the Lock Manager's functions in the cluster.

Virtual Balance Set Statistics

Rates can be reported for bytes read, bytes written, real and virtual transitions, map buffer allocations, real slot availability, virtual selection failure rates, virtual map hit rates, map count, fluid balance, recopy rate, and Virtual Balance Set CPU time.

DEC Distributed Transaction Manager Statistics

DDTM is the protocol used for two-phase commits by RMS, Oracle Rdb and Oracle DBMS. As of OpenVMS v7.3.1, HP has also documented it for public use. SightLine can report on DDTM rates for start, prepare, abort, end, 1-phase commits, remote branch and remote add branch as well as a range of transaction lengths.

Uptime Statistics

SightLine will report on total uptime since the last boot or uptime for the current month either on a 24-hour basis for the full week or divided according to a schedule of selected time periods.

Galaxy Events

For systems that are instances in an OpenVMS Galaxy, SightLine can provide notification of various Galaxy-related events, such as other instances joining and leaving the Galaxy, CPUs becoming active or inactive in the instance, CPUs joining or leaving the configure set for the instance, updates to the Galaxy configuration tree, modifications to CPU I/O preferences, and time differential changes.

Galaxy Statistics

For ongoing data pertaining to a system which is an instance in an OpenVMS Galaxy, SightLine can report shared memory statistics (total, used, free, bad and CPP count), CPUs active, made active, made inactive, added to the configure set, leaving the instance, instances joined, Instances left, Tree updates, potential CPUs, number of times the

Galaxy has been incarnated, as well as, identification information regarding the particular Galaxy member.

Contact Information

SightLine Systems Corporation
11130 Fairfax Boulevard
Suite 200
Fairfax, VA 22030
(703) 563-3000
sales@sightlinesystems.com

OpenVMS Technical Journal V13

Adding Physical CD Support to the SIMH VAX

Michael D. Duffy

Introduction

As aging hardware keeps chugging along, maintenance costs continue to rise and replacement parts become more and more scarce. In some cases, that ancient MicroVAX is still running a critical application that has never been ported to another platform. But since the VAX never crashes and always comes back up after a power failure, nobody seems too concerned that everyone who understood the application left the company years ago. In the back of your mind, you know you should really do something about that.

Some sites choose to keep the application intact but move it to an emulated VAX running on some other system, usually at a substantial performance increase and cost reduction. Commercial emulators can be quite expensive, but open source solutions may not have all the features required to get the most out of the product.

In response to these concerns, Migration Specialties asked me if a copy of OpenVMS running under SIMH could be allowed direct access of a CDROM device on the host computer, whether it was running OpenVMS or Windows. (UNIX systems need no such enhancement, as pointing a container file specification to /dev/cdrom works with the standard SIMH version.)

SIMH is a hardware simulator that can act as any one of more than twenty different machines, but I was asked to concentrate on the VAX 3900 simulator, as it was the most likely existing configuration to be used by the customer sites we envisioned.

This article explores the challenges and solutions found by the author while adding Host CD support to the SIMH open source VAX emulator. The solution demonstrates that a developer may not need a wealth of experience with SIMH in order to add significant functionality to it, and may serve as a template for your own enhancement ideas.

The Plan

SIMH disk I/O is based on container files representing disk drives, a standard approach among many different emulators. A file on a host device acts as a repository for a bit-for-bit copy of the entire contents of a disk volume attached to the computer to be simulated. Previously, a user could create a disk image file from the contents of a physical CD and then attach the image file to the simulator. Our goal was to eliminate this time-consuming step and allow access directly to the host CD drive.

The problems I anticipated at the outset were the lengthy delays inherent in CD access and how to support OpenVMS mount verification and CD volume switching between the emulated environment and the host.

SIMH disk I/O is synchronous, that is, the simulator stops and waits for container file reads and writes before continuing. This is true whether or not the data are immediately delivered to the guest OS. Some experimentation would be necessary to determine how

tolerant OpenVMS would be of these delays, and whether an asynchronous delivery mechanism would be needed.

Experimentation with CD volume changes and guest MOUNT and DISMOUNT activities would be needed to determine 1) whether there is a simple, reliable way to tell within SIMH itself when the guest has mounted or dismounted a volume and 2) whether allowing a host I/O to fail is sufficient to trigger OpenVMS mount verification on the guest side.

Proof of Concept

I decided to put together the simplest solution to probe these areas. Intercepting what SIMH thought would be a container file read and redirecting it to a CD device seemed like the obvious choice. After reading the data, I would insert it into the same buffer that would have been used for container file data and allow SIMH to proceed from there. How hard could it be?

The first step was to determine where the I/O should be intercepted and how to determine which of the I/Os passing through that point were the right ones to redirect. Routine sim_fread() in source file SIM_FIO was chosen because all container file reads pass through it. Since writes were to be disallowed, placing the test in the read dispatcher would automatically prevent writes from being attempted by any new code I would write. Later testing indicated an improvement might be made possible by moving the test elsewhere, which will be discussed later.

One of the arguments to sim_fread() is a unique file identifier, called fptr, that could be used to identify the I/Os of interest. I quickly decided that for the proof of concept, I would open a container file as normal when SIMH was initialized, but intercept the I/Os to that file at runtime and send them to the CD instead. Later, after proving the concept, I would remove the container file or at least make it into a tiny stub file that did not consume space on the host disk.

The VAX implementation in SIMH uses some of the same code as the PDP11 for I/O. Specifically the RQ* controllers and disk types are shared via the PDP11_RQ* code found in SIMH's PDP11 source directory. I looked through the code defining device types and the data structures that represent them in SIMH, finding that due to the number of bits representing the device type and the number of devices already supported, there were no available slots for adding a new device type. I therefore used the slot that had previously been taken by the RA70 device type, the last device in the list, and also one of the smaller disk models. I renamed that device PHYCD and copied the remaining values from the RRD40 device definition already present in SIMH. I then created a Physical CD Control Block to store various information related to the PHYCD device.

The next step was to modify the SET and ATTACH commands to recognize references to the PHYCD type. SET and ATTACH are used when SIMH is started, to define the devices that will be present on the simulated system and associate them with container files, respectively. I modified the ATTACH code to open a stub container file and store

its file pointer in the PHYCD control block, so that Physical CD I/Os passing through sim_fread() could be identified via the fptr argument. I also changed ATTACH to treat the container file argument as a device specification when the device type is PHYCD. The new syntax is compared with the existing method below.

Traditional container file examples:

SET RQ1 RRD40
ATTACH RQ1 DKA0:[SIMH]CD_IMAGE_FILE.DAT

New Physical CD access examples:

SET RQ1 PHYCD
ATTACH RQ1 DKB400: (OpenVMS host)

This example causes SIMH device RQ1 to be attached to physical CD device DKB400: on the host system.

Now it was time to actually redirect an I/O to the CD drive once it was detected, but the information passed to sim_fread was still incomplete: The desired length of the read is present, but the starting position is not. SIMH sets the container file read offset via an fseek() before sim_fread() is called, so I added a similar test at the fseek() to record the desired starting position in the Physical CD control block.

Since the guest OpenVMS always starts reads on 512-byte boundaries, I did not have to make any adjustments to the starting position or length of read on an OpenVMS host. On a Windows host, however, reads start on 2048-byte boundaries, so some calculations must be performed to read the proper block and extract the portion the OpenVMS guest requested (and introduce an opportunity for a small read-ahead cache, since the data would otherwise be wasted.)

These values were passed to a $QIOW to read the CD and place the data into the buffer ordinarily used for the container file.

For the purposes of initial testing, I added a $MOUNT system service at SIMH startup to ensure the CD device would be available. For now, no provision for removing or switching CDs was included. That would come later.

Initial Testing

Once this bare-bones approach was ready, I booted up a copy of OpenVMS/VAX V7.3 with the PHYCD unit attached to device DUA1: and placed a CD in the host drive.

As I anticipated, there were delays of a few seconds whenever the CD needed to spin up, but the guest copy of OpenVMS was very tolerant of these delays, with no failures or

errors noticed to date. A given application might be less forgiving, but OpenVMS itself seems not to mind very much.

Once the CD was up to speed, further reads proceed normally, with no effect on the simulated system that would be obvious to a human. However, overall system throughput is reduced during the time the CD is being actively read. This is also true of container files, but is magnified somewhat by the lower performance of CD drives.

It was decided to expand this concept to include support for switching CD volumes, handling errors and triggering OpenVMS mount verification within the guest OS at the right times.

Supporting CD Swapping and Removal (OpenVMS)

My initial thoughts were that the OpenVMS and Windows-host version would differ greatly with regard to handling the user removing or switching the CD in the drive. The two versions turned out to resemble one another much more than I originally thought, due to an error I made early in development and didn't catch until a significant amount of code had been written. The reader can use my mistake to avoid doing extra work on any similar project.

My first attempt at mount verification support on OpenVMS hosts was based on this general idea:
I planned to detect mount verification on the host and simply pass it into the emulated environment. First, I would issue the I/O with a timeout event flag. If the I/O timed out, I would check to see if the host CD is in mount verification. If so, I would cancel the I/O, perform an internal DISMOUNT/ABORT and Mount the CD again. Then I would reissue the I/O, while simultaneously notifying the guest operating system that mount verification should be triggered.

It was at this last step where I made a slight mistake by writing all the other code first before carefully examining the mechanism by which I would notify the guest that something unusual had happened. As it turned out, the code worked well, but by the time it was triggered, it was already too late to notify the guest OS.

Had I checked in a little more depth, I would have found that the opportunity to signal mount verification (which means to return an offline status for the emulated device) had already passed. Routine rq_rw_valid(), a series of validity checks that gets performed before the I/O is started, is the right place to do this, but is already finished before the I/O is sent on its way.

In order to get the solution working more quickly, I made the OpenVMS version behave more like the Windows version (already underway), but left the aforementioned code in place with an eye toward fixing it at a later date. I envision doing the read preemptively in rq_rw_valid() and leaving the data in a buffer that can be retrieved later in sim_fread(). In this way, timely mount verification alerts can happen, while avoiding extra steps.

Supporting CD Swapping and Removal (Windows)

How, then, to support OpenVMS-style mount verification on other than an OpenVMS host? Some mechanism was required to detect when a different CD appeared in the drive. I decided to copy the first 2KB of the drive "from time to time," the exact meaning of which had yet to be determined. Then, when a different CD appeared in the drive, a different 2K of data would be found in the first block, and SIMH could return "offline" to the client while simultaneously updating the cached copy of the new data. SIMH could then continue normally, while the guest OS began its mount verification processing.

In order to have the volume checked on a regular basis, I decided to close the handle to the host drive after a few seconds of inactivity. Any period of time long enough to physically switch the CD would trigger this condition, after which SIMH would detect the new CD.

Regularly opening and closing the handle to the drive does add some overhead, but the delay happens at the same time the CD is coming up to speed, and so only slightly lengthens an already noticeable delay.

Once this mechanism was working, I modified the OpenVMS version to do the equivalent opening and closing via $DISMOU and $MOUNT. It works as well for OpenVMS as it does for Windows, but the OpenVMS version also contains the code described in the OpenVMS-specific section, which a future version can take advantage of.

Both the OpenVMS and Windows host version also return "offline" if the CD drive contains no CD, causing the guest OS to behave as expected. OpenVMS returns "medium is offline" if a MOUNT is attempted, or retries once per second when mount verification is underway.

Room for Improvement

The code described above represents the first attempt at supporting physical CD access within SIMH. There are a couple of obvious improvements that can and will be made as time allows.

First, the inactivity timer test is currently located in the VAX CPU instruction dispatcher. This was a convenient and reliable place to put it during testing, but is not ideal. Ideally, a standard SIMH timer event should be used, which will reduce overhead.

Secondly, the OpenVMS version could benefit from placing a read into rq_rw_valid() for the reasons described above.

Finally, the code as currently designed sometimes triggers an "offline" signal when it is not necessary to do so. Any time a CD volume is switched (and also when SIMH is started with an empty CD tray), the "First-2K test" will trigger. Mainly, this is because I

am currently unaware of a good way to tell when the OpenVMS client has dismounted a volume. From SIMH's perspective, I/O simply stops for some period of time and resumes later. If SIMH could reliably know when a DISMOUNT has occurred, First-2K testing could be suppressed for the next sequence of I/Os.

Possible ways to improve this behavior include detecting a guest MOUNT, perhaps by recognizing the specific block number(s) requested, or by detecting a DISMOUNT/UNLOAD by some condition recognizable from the PDP11_RQ code. Currently, a spurious "medium is offline" or transient mount verification cycle remain as the only obvious bug in the new functionality. It should be noted that the guest OS can simply retry the operation and everything will work itself out, but it would be preferable to remove this behavior.

For More Information

You can find the SIMH base source code at http://simh.trailing-edge.com and the source code and installation kits for the enhanced SIMH functionality at http://www.MigrationSpecialties.com.

To subscribe to the SIMH mailing list, send a SUBSCRIBE message to simh@trailing-edge.com.