OpenVMS Technical Journal V8

System Service Interception

Ruth Goldenberg

Overview

Many of the actions that the OpenVMS operating system takes on behalf of a user are implemented as procedures called system services. A user application requests system services directly, and components such as the file system also request system services on an application's behalf. For example, a program calls the Get Job/Process Information ($GETJPI) system service to find out specific details about another process or job and the Queue I/O Request system service to perform input from or output to a specific device.

System service interception (SSI) is a mechanism that enables system services to be intercepted and user-specified code to run before, after, or instead of the intercepted service. An image can declare routines to perform pre-service processing, post-service processing, system service replacement routines, or any combination thereof.

The implementation of the mechanism limits interception to system services in executive images; that is, system services in privileged shareable images cannot be intercepted.

SSI is used by the Debugger and OpenVMS tools such as the Heap Analyzer and Performance Coverage Analyzer (PCA). The Debugger, for example, uses it to implement global section watchpoints and speed up static watchpoints. The Heap Analyzer needs to follow changes to memory layout. It intercepts memory-changing services ($EXPREG, $CRMPSC, and so on) to track and display these changes as they occur. At one customer's site, SSI has been used to intercept file opens and closes done by a third-party application to minimize unnecessary operations: $CLOSE is intercepted to keep a file open and $OPEN to refrain from re-opening a still-open file.

SSI has been available, but not previously documented, on OpenVMS Alpha since Version 6.1. It is available for OpenVMS I64 with Version 8.3 and later versions.

This article first describes how OpenVMS dispatches to executive system services on both Alpha and I64 platforms and then how these services are intercepted.

System Service Dispatching

Most system service procedures are contained in executive images and reside in system space; others are contained in privileged shareable images.

System services typically execute in kernel or executive access mode so that they can read and write data structures protected from access by outer modes.

The implementation of inner mode system services is based on a controlled change of access mode. Although an unprivileged process can enter an inner access mode to execute code in that mode, it can execute only procedures that are part of the executive or that have been specifically installed by the system manager.

When a program requests an inner mode system service, it executes a system service transfer routine that serves as a bridge between the requestor's mode and the inner mode in which the service procedure executes. System service transfer routine names are resolved using the same mechanisms as externally visible names in a shareable image.

The following sections describe how executive system service names are resolved and how control is transferred to an executive system service procedure on both OpenVMS Alpha and I64 systems.

System Service Dispatching on OpenVMS Alpha

Every OpenVMS Alpha procedure is described by a data structure called a procedure descriptor (PD), which contains the address of the procedure's code entry point and information about its type and characteristics. This is true for every procedure, whether it is a procedure in an executable, shareable, or executive image, or whether it is an ordinary procedure, a system service transfer routine, or a system service procedure.

A compiler generates a PD for each procedure in a module and places them together in a program section called a linkage section.

A linkage section also contains information about calls to external procedures. A call to an external procedure is represented as a two-quadword data structure called a linkage pair. By the time a call using a linkage pair is executed, the first quadword of the linkage pair must contain the external procedure's code entry address, and the second quadword must contain the address of its PD.

The global symbols from object modules in a shareable image that are to be visible externally are called universal symbols. Each universal symbol in an image is represented within two image structures: the global symbol table and the symbol vector. A global symbol table entry gives the offset of the corresponding symbol vector entry. A symbol vector entry consists of two quadwords. For a universal symbol that is the name of a procedure, the two quadwords hold addresses of the procedure's code entry point and its PD. That is, they form replacement contents for a linkage pair.

When the linker links an object module containing a call to an external procedure in a shareable image, the linker cannot resolve the contents of the linkage pair because a shareable image is typically not assigned address space until it is activated. Instead, the linker records in the image it is building the need for the image activator to resolve the contents.

When an image and a shareable image with which it is linked are activated, each procedure symbol vector entry in the shareable image is updated to contain the actual addresses of the PD and code entry point for that procedure. The image activator resolves the linkage pair for the call into the shareable image by replacing the linkage pair's contents with the corresponding contents from the shareable image symbol vector entry.

To call the procedure in the shareable image, compiler-generated code in the main image loads from the linkage pair the target procedure's PD and code entry addresses. It transfers control to the code entry address, saving the return address in a register.

In general, transferring control to an executive system service resembles transferring to a procedure in another image. It differs in the following ways:

- There is one symbol vector, built as part of the SYS$PUBLIC_VECTORS.EXE image, for all system services in all executive images. The image also contains a global symbol table listing all the system service transfer routine names and their offsets in the image's symbol vector.
- The PDs for all executive services initially describe system service transfer routines within SYS$PUBLIC_VECTORS.EXE. Its symbol vector initially contains linkage pairs with the addresses of these PDs and transfer routines.

When an executive image containing a system service is loaded during system initialization, information about the service in SYS$PUBLIC_VECTORS.EXE is modified.

- If the service is a mode of caller service, the symbol vector entry contents are modified to reflect the location of the service PD and entry point in the executive image. In other words, the system service call goes directly from the caller's image to the service procedure.
- If the service is an inner mode service, the transfer routine is modified to identify the service and specify its mode.

Figure 1 shows a user image that calls both an inner mode (SYS$k) and a mode of caller (SYS$moc) system service. When an image that requests a system service is linked, the linker resolves the system service transfer routine name by searching the SYS$PUBLIC_VECTORS.EXE global symbol table. It stores the symbol vector offset corresponding to the system service transfer routine in the linkage section of the image making the service request. It stores information about the need to update that linkage pair in the fixup section of the image.

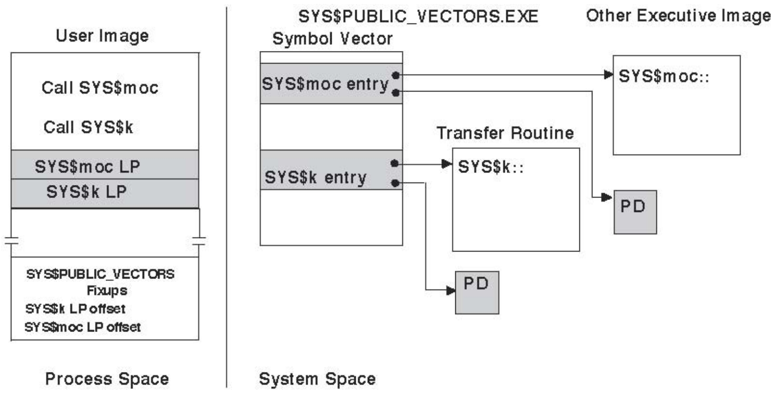

Figure 1 - Resolving OpenVMS Alpha System Services Names

In the figure, the linkage section has a shaded linkage pair for each of the two services. Its fixup section lists the two corresponding fixups.

Unlike other shareable images, SYS$PUBLIC_VECTORS.EXE is not activated with an executable image. Instead, as part of the executive, it is loaded into system space during system initialization. Because all processes map system space, SYS$PUBLIC_VECTORS.EXE is shared.

In the SYS$PUBLIC_VECTORS.EXE symbol vector in Figure 1, the two entries for these services are shaded. The entry for SYS$moc points to a PD and an entry point within the executive image that contains that service. The entry for SYS$k points to the PD and a transfer routine within SYS$PUBLIC_VECTORS.

When an executive image with an inner mode service is loaded, the corresponding transfer routine is modified to contain an instruction that loads a service-specific value into R0 and either a CALL_PAL CHMK or a CALL_PAL CHME instruction.

Figure 2 shows a flow to and from the example kernel mode service SYS$k.

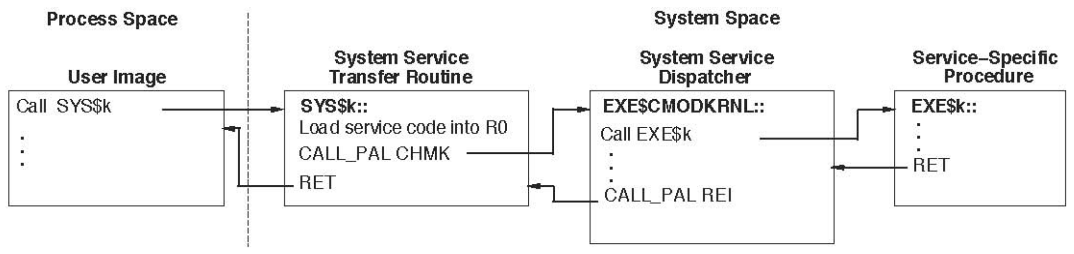

Figure 2 - Transferring to Inner Mode OpenVMS Alpha System Services

The user image calls the system service, passing control to the transfer routine in SYS$PUBLIC_VECTORS.EXE.

Its CALL_PAL CHMK instruction causes an exception and a mode change to kernel. The change mode exception is handled by the system service dispatcher, which uses the value in R0 to determine which system service procedure to call.

System Service Dispatching on OpenVMS I64

An OpenVMS I64 procedure can be described by a data structure called a function descriptor (FD), which contains the address of the procedure's entry point and the address that should be loaded into its global pointer (gp) register. The gp is a base address for references to the short data segment, which includes small writable data, pointers to external data, and procedure linkages.

An FD roughly corresponds to an Alpha linkage pair and is accessed in transferring control from one procedure to another. Most FDs are created by the linker. Each externally visible procedure has one so-called "official" FD, but other procedures do not necessarily have any.

To describe universal symbols in a shareable image, the OpenVMS I64 linker creates a symbol vector and the equivalent of a global symbol table. The global symbol table lists the shareable image's universal symbols, their symbol vector indices, and their types. A symbol vector entry contains one quadword for each universal symbol; in the case of a universal procedure it contains an FD address.

A linkage to a procedure in another image or another image segment in the same image is represented in the calling procedure's short data by a local FD or a pointer to an official FD. By the time a call using the FD is executed, it must contain the target procedure's entry point and the value for its gp.

If the caller and target procedure are in different images, the linker generates an FD, a procedure linkage table (PLT) routine in the same program section as the call site, and a br.call to the PLT routine.The addresses in the local FD are fixup targets. The PLT routine uses the FD to load the gp register and branches to the target procedure's entry point.

In general, transferring control to an executive system service resembles transferring to a routine in another image. It differs in that SYS$PUBLIC_VECTORS.EXE contains the symbol vector for system services in all executive images and the FDs for all such services. Its FDs point either to executive images or to service-specific routines in a system space promote area: FDs for mode of caller services point to service-specific routines in executive images; FDs for inner mode services point to service-specific routines in a part of system space called the promote area.

Figure 3 shows an I64 image that calls both an inner mode (SYS$k) and a mode of caller (SYS$moc) system service. Note that the figure is somewhat simplified; Section *System Service Interception on OpenVMS I64* contains a more complete description.

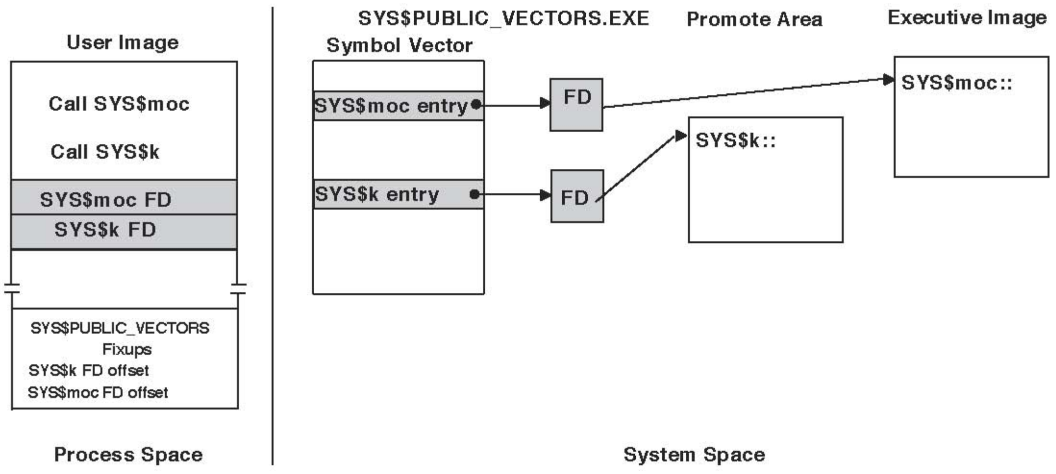

Figure 3 - Resolving OpenVMS I64 System Services Names Against the Original SYS$PUBLIC_VECTORS.EXE

On I64 platforms, the mechanism for changing to an inner access mode is to execute an epc instruction on a page with a special protection code. Such a page is called a promote page. A set of virtually adjacent promote pages is called a promote area. Routines in the promote area change access mode and transfer control to the system service dispatcher.

When an executive image is loaded that contains an inner mode service, a transfer routine specific to that service is created in the promote area. The SYS$PUBLIC_VECTORS.EXE FD for the inner mode service is modified to contain the address of the promote area transfer routine.

Figure 4 shows a simplified flow to and from the example kernel mode service, SYS$k.

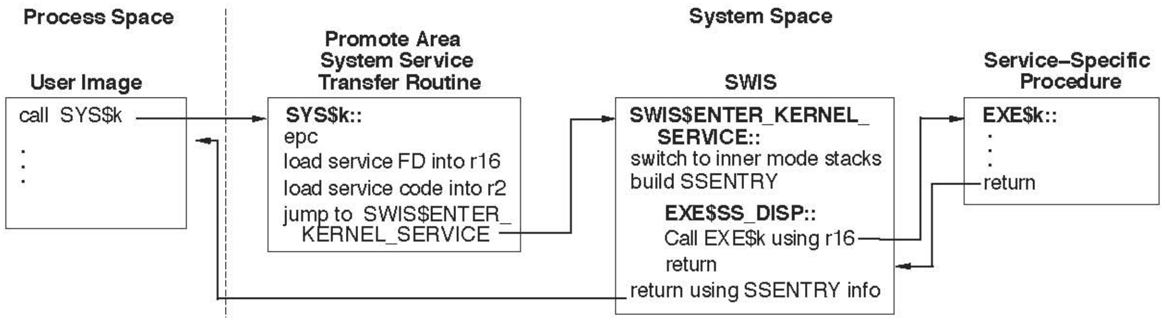
Process Space
System Space
User Image
call SYS$k
Promote Area System Service Transfer Routine
SYS$k::
epc
load service FD into r16
load service code into r2
jump to SWIS$ENTER_KERNEL_SERVICE
SWIS
SWIS$ENTER_KERNEL_SERVICE::
switch to inner mode stacks
build SSENTRY
EXE$SS_DISP::
Call EXE$k using r16
return
return using SSENTRY info
Service–Specific Procedure
EXE$k::
return

Figure 4 - Transferring to Inner Mode OpenVMS I64 System Services

The user image calls the system service, passing control to the transfer routine in the promote area.

The promote area routine executes an epc instruction to change mode to kernel, loads the service's FD address into a register, loads a code identifying the service into another register, and branches to an executive component called software interrupt support (SWIS).

SWIS switches to the inner mode stacks (memory and register). It builds a stack data structure called an SSENTRY that records the state of the thread of execution at the time of the mode switch, including the address to which control should be returned. It then transfers to the system service dispatcher, which uses the service FD to load the gp for the system service procedure and to call its entry point.

When the service is done, the system service dispatcher returns to SWIS. SWIS restores state from the SSENTRY, switches back to the outer mode stacks, and returns to the system service caller.

System Service Interception (SSI)

SSI enables system services to be intercepted and user-specified code to run before, after, or instead of an intercepted service. When a system service request is intercepted, SSI code saves the complete system service argument list in a VAX-style argument list format and calls any pre-processing routine. It then calls the real service or the replacement routine using the saved argument list. When the service replacement routine returns, SSI code calls any post-processing routine with status from the service or replacement routine.

SSI is local to the process that has enabled it.

Although the same SSI application programming interfaces (APIs) are provided on both Alpha and I64 platforms, the underlying SSI implementation is quite different. SSI requires alterations in either the system service name resolution or the dispatch to system services. On OpenVMS Alpha, the SSI implementation depends on altering service name resolution. On OpenVMS I64, the implementation depends on altering transfer routines. The following sections describe these two different implementations.

System Service Interception on OpenVMS Alpha

On OpenVMS Alpha, the SSI mechanism is implemented in a privileged shareable image called SYS$SSISHR.EXE. Its symbol vector is laid out in the same order as the symbol vector for SYS$PUBLIC_VECTORS.EXE. SYS$SSISHR.EXE essentially provides a set of jacket routines for all the SYS$PUBLIC_VECTOR services as well as a set of APIs to declare pre- and post-processing routines and replacement routines.

If the main image was linked with SYS$SSISHR.EXE or linked /DEBUG, the image activator activates SYS$SSISHR.EXE before doing any fixups. Once it has been activated, the image activator fixes up references to SYS$PUBLIC_VECTORS.EXE symbols using the SYS$SSISHR.EXE symbol vector instead. This means that services can be intercepted only from images activated with and after SYS$SSISHR.EXE.

If the main image was not linked with SYS$SSISHR.EXE and was not linked /DEBUG, its references to system service names are fixed up against SYS$PUBLIC_VECTORS.EXE. A subsequent dynamic activation of DEBUG with the DCL CTRL/Y $DEBUG sequence or with connect/disconnect cannot cause SYS$SSISHR.EXE to become activated in time to intercept system service calls from images already activated because they have already been fixed up.

Figure 5 shows the image of Figure 1 with references to SYS$PUBLIC_VECTORS.EXE symbols fixed up against SYS$SSISHR.EXE: the shaded LPs get the contents of the related SYS$SSISHR.EXE symbol vector entries and thus cause transfer to SYS$SSISHR.EXE. All of the symbol vector entries transfer to the same place, a routine called SSI_TRANSFER. SSI_TRANSFER is, however, entered with the address of the PD unique to that system service and thus can determine which service was requested.

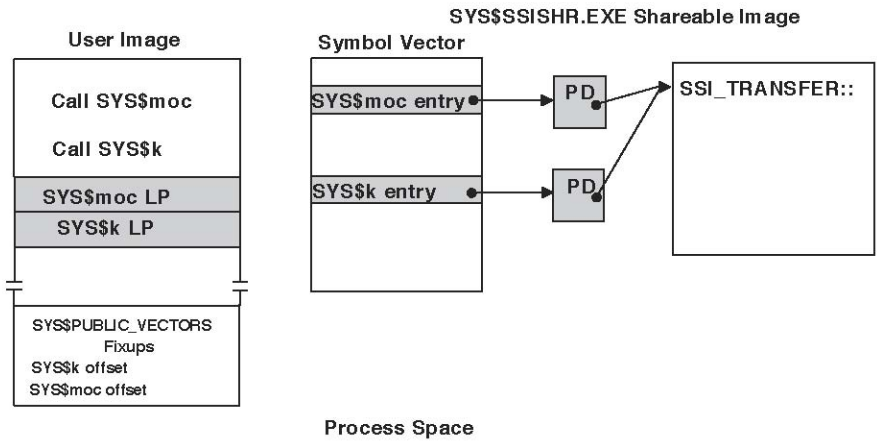

Figure 5 - Intercepting OpenVMS Alpha System Services

SSI_TRANSFER is written in assembly language to enable direct access to registers involved in the service call. It saves the complete system service argument list on the stack in VAX-style argument list format. It then calls the main SSI routine, SSI_MAIN_TRANSFER, with the following arguments:

- The address of the VAX-style argument list
- The service caller's return address
- The address of the service PD within SYS$SSISHR.EXE

SSI_MAIN_TRANSFER transforms the service PD address to the address of the service PD within SYS$PUBLIC_VECTORS. It determines whether any pre-processing, post-processing, or replacement routines have been declared and calls them before, after, and/or instead of the actual system service. It takes the following steps:

1. It determines whether the service was requested from an inner mode. If so, it calls the real system service with the saved arguments.
2. It determines whether the service caller was SYS$SSISHR.EXE itself. If so, it calls the real system service with the saved arguments and returns.

3. It copies the lists of pre-processing and post-processing routines in case any pre-processing or post-processing routine should declare or cancel a pre-processing or post-processing routine. Any such call will not take effect until the next system service request.
4. If any pre-processing routines have been declared, it calls them in the reverse order in which they were declared.
5. If a replacement routine was declared for this service, SSI_MAIN_TRANSFER calls it. Otherwise, it calls the actual system service.
6. When the replacement routine or actual system service returns, R0 contains the status value. SSI_MAIN_TRANSFER saves R0 to pass it as an argument to post-processing routines.
7. If any post-processing routines have been declared, it calls them in the order in which they were declared.
8. It restores R0 as the status value and returns to the caller.

System Service Interception on OpenVMS I64

On I64, the SSI mechanism is implemented in a shareable image called SYS$SSISHR.EXE and a privileged shareable image called SYS$SSISHRP.EXE. SYS$SSISHR.EXE provides the APIs to declare and cancel pre- and post-processing routines and replacement routines. SYS$SSISHRP.EXE contains inner mode support routines for SYS$SSISHR.EXE.

A system space copy of SYS$PUBLIC_VECTORS.EXE created during system initialization provides hooks for interception. References to system services from within executive images are fixed up against the original version of SYS$PUBLIC_VECTORS.EXE. By default, references to system services from all other images are fixed up against the copy of SYS$PUBLIC_VECTORS.EXE.

When a process is created, the physical pages that make up the system-space promote area are double-mapped into its P2 space. Every FD in the copy of SYS$PUBLIC_VECTORS.EXE, even one for a mode of caller service, points to a service-specific routine in the P2 space mapping of the promote area. Figure 6 shows an image fixed up typically, namely against the copy of SYS$PUBLIC_VECTORS.EXE.

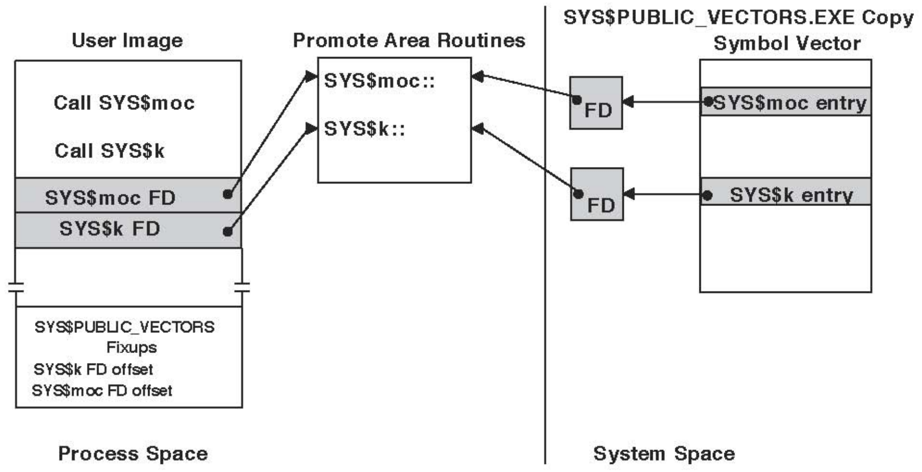

Figure 6 - Resolving OpenVMS I64 System Services Names Against the Copy of SYS$PUBLIC_VECTORS.EXE

When an image calls an SSI API to request system service interception, a process-private copy of the promote area is created, and promote area routines are modified to transfer control to SYS$SSISHR.EXE.

Figure 7 shows an image set up to intercept SYS$k on OpenVMS I64.

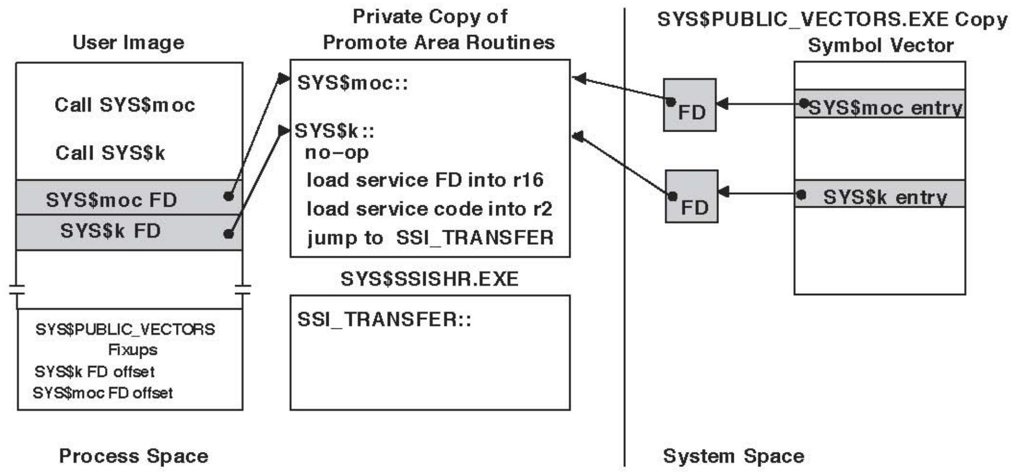

Figure 7 - Intercepting OpenVMS I64 System Services

The modified transfer routines for both mode of caller and inner mode services record the address of the system service FD and a code that identifies the service and transfer to SSI_TRANSFER, in SYS$SSISHR.EXE.

As on Alpha, the I64 version of SSI_TRANSFER is written in assembly language to enable direct access to registers involved in the service call. It saves the complete system service argument list on the stack in VAX-style argument list format. It then calls the main SSI routine, SSI_MAIN_TRANSFER, which is common code with the OpenVMS Alpha version.

As a result of this implementation, services in a process can be intercepted regardless of when SYS$SSISHR.EXE and SYS$SSISHRP.EXE are activated. Once system service interception has been enabled in a process, any SYS$PUBLIC_VECTORS service requests made from user mode can be intercepted. This includes service requests made from shareable images linked with the main image.

For more information

For a detailed description of image activation, see OpenVMS *Alpha Internals and Data Structures: Scheduling and Process Control*, available from Digital Press.

For a detailed description of system service dispatching on OpenVMS Alpha, see *OpenVMS AXP Internals and Data Structures Version 1.5,* Digital Press

For a detailed description of the OpenVMS Alpha and I64 calling standards (e.g., PDs, linkage pairs, and FDs), see the *HP OpenVMS Calling Standard*, available at http://h71000.www7.hp.com/doc/82final/5973/5973PRO.HTML

For information on the use of SSI, please check the HP OpenVMS Systems Documentation home page for forthcoming documentation.

For information on a mechanism to intercept privileged shareable image services, see *Faking it with OpenVMS Shareable Images* in Volume 7 of the OpenVMS Technical Journal http://h71000.www7.hp.com/openvms/journal/v7/faking_it_with_openvms_shareable_images.html.