A Robust VMS LOGOUT Driver Activator

Larry L. Johnson
Texas Instruments
McKinney, Texas

ABSTRACT

We have constructed a data-flow model of a generic software product in the context of a general time-sharing environment. In this model, points of control for each product are defined for each "system-event". One of those events is the deletion of a process (specifically, LOGOUT). The current implementation of the activation of our logout driver (which calls per-product logout functions) is weak. An unprivileged user can exit the system in a manner which will bypass the activation of the logout driver. By the definition of our model, logout operations are not optional. A weak activation of the logout driver is intolerable. This study was undertaken to determine the feasibility of implementing a robust logout driver activation.

This study established the feasibility of implementing a robust logout driver activation by:

1. proposing mechanisms on which a solid logout driver activation can be based.
2. producing a functional prototype based on a selection of these mechanisms.
3. proposing the mechanism and general structure of a production version.
4. outlining the course of action to produce such a version.

A comparison of the advantages and disadvantages of various available mechanisms is presented. Based on the experience derived from the mechanisms for the prototype, recommendations are made for the selection of mechanisms and structure for an eventual production version.

The considerations to be addressed by future efforts is clearly and specifically defined. The major thrust of the considerations is to evolve the current functional prototype in order to meet design goals of per-site flexibility, staying out of the way of per-machine system programming, and providing the per-machine system programmer with tools to interface with the mechanism more effectively than if the programmer attempted to accomplish his task with "vanilla" VMS.

2 Problem Analysis

2.1 Primary Objective

The primary objective of the study is to determine and eventually implement a usable mechanism to force the execution of a system-wide logout command file.

The mechanism is to be hardy and non-defeatable by an unprivileged VMS user.

2.2 Decomposition of Primary Objective

The only common point of exit for processes of all types is through the VMS image rundown. Process deletion is a special, extended case of image rundown in VMS systems. Therefore the first objective of the study must be to intercept image rundown, and recognize the case of process deletion (rundown of the process).

Having intercepted process deletion, the process must be restored to a state capable of defeating user control and forcing the execution of a normal DCL command file in a normal user process context.

Having established a suitable environment, a mechanism is required to force DCL to execute a system-wide logout command procedure. (Simulate the "@command-file" behavior).

The command file must have a mechanism by which it can request a process deletion (without forcing its own execution once again).

The procedure must be immediately available to the process context, which means it must be permanently mapped and not suffer the effects of image rundown.

A method for loading such an image must be developed.

In summary, to achieve a solution the following subproblems must be solved:

1. Design mechanism to intercept normal VMS process deletion. (special case of image rundown interception).
2. Design mechanism to recover process from deletion.
3. Design mechanism to transfer control from user to system-wide logout command procedure.
4. Select rundown interceptor image residency.
5. Develop prototype rundown interceptor using above mechanisms.
6. Develop procedure to load rundown interceptor.

3 Discussion of Technical Issues and Candidate Solutions

The user is presumed to be intimately familiar with VMS Internals for this material.

3.1 Process Deletion Interception

In order to intercept process deletion for all processes, control must be gained each time VMS deletes a process. VMS supplies the hooks necessary for this through a dispatch vector mechanism described below.

3.1.1 VMS Dispatch Vectors.

VMS provides for the call of user-written routines at points critical in the VMS architecture: For each of these critical points VMS maintains two hooks:

1. System-wide vector pointer (S0 space).
2. Per-process vector pointer (P1 pointer page).

The critical points at which control is provided for each of the hooks are:

1. Change mode to kernel dispatching. If the VMS change mode to kernel dispatcher does not recognize a system service code, it will call user-written dispatchers through the system-wide pointer if it is non-zero. If the system-wide dispatcher does not recognize and handle the change mode code, user-written dispatchers are called through the per-process pointer.
2. Change mode to executive dispatching. Dispatching is handled analogously to the kernel case.
3. Image rundown dispatching (process deletion is a special case of image rundown under VMS). The per-process dispatch vector is called through the per-process pointer. If the system-wide dispatch pointer exists (is non-zero) it is called. The vectors are called exclusively by SYS$RUNDWN and by SYS$DELPRC.

The following are the symbolic names of the locations of each of these pointers.

	System-wide (S0 Space)	Per-process (P1 Space)
Change mode kernel	EXE$GL_USRCHMK	CTL$GL_USRCHMK
Change mode executive	EXE$GL_USRCHME	CTL$GL_USRCHME
Image Rundown	EXE$GL_USRUNDWN	CTL$GL_USRUNDWN

The system-wide pointers are not set by VMS. There is no established protocol for their use. There is, therefore, no chance of conflicting with VMS operations in the use of these vectors. However the vectors are truly system-wide and will be invoked by any process. Any exclusions would have to be dismissed by analyzing the calling environment at each invocation.

The per-process vectors are used by the image activator in mapping privileged shareable images. This is the supported method for providing user written change mode dispatchers and rundown services on a per-image basis.

The per-process pointers point to vector tables in P1 space. These are referred to as VMS dispatch vectors. The tables are one-half page each, located in two contiguous pages. There is a fourth set of pointers/vectors for message sections. The fourth table does not follow the format of the other three described here.

The tables are not strictly "jump tables", but mini-routines composed of JSB instructions to each privileged dispatcher and terminating with an RSB instruction. The image activator adds a JSB instruction to each of the tables as required in the encounter of any installed privileged shareable image. The number of such vectors is restricted by the size of the dispatch area (256 bytes for each table... 42 JSB sequences).

The image activator keeps track of the original (permanent) size of each vector table and, at image rundown, over-writes the JSB op-code written there at image activation with the original RSB op-code. Thus, at image rundown the vector table is reset to look as it did before image activation.

It is possible in VMS V4.x systems to make these vectors permanent by updating the location containing the original size of the vector table to the size of the table with the permanent vectors in place. (This possibility did not exist under V3.x and earlier systems. The vector tables in those systems were reset to no vectors at each image rundown.)

3.1.2 Selection of Vector Mechanism

The use of the system-wide vectors offers the following advantages.

1. No VMS established protocol. There is little chance of clashing with VMS's use of the vector.

2. Per-process modifications are not necessary. Once established, the rundown intercptor will be activated for every rundown in the context of every process, regardless of its environment.

The disadvantages of system-wide vectors are:

1. A protocol for use of the vector would have to be established. VMS's use of the per-process vector could serve as a convenient model for the protocol, making the effort straight-forward.

2. Since a logout command procedure driver is meaningful only to those processes which map DCL, ineligible processes would have to be detected so no operation would be performed, or an alternate image-based system-wide logout be performed.

3. The use of the system wide vector would require mapping the image into S0 rather than P1 space, since each P1 space may have the image mapped into different address ranges. P1 space could be used to keep per-process data. This would require a check on each activation to assure that the P1 data area has been initialized, and the initialization performed if it has not.

   Per-process S0 allocated areas could also be used, established on any access indicating absence of structure. This approach essentially implements an extended PCB and should be accessed with similar protocol.

The per-process vectors offer the following advantages:

1. They can be loaded on per-process basis by the system-wide login procedure as desired. If the image is activated, it is eligible to run. Fewer environmental determinations are necessary.

2. Any P1 data area can be mapped at the same time as the rundown image.

The per-process vectors offer the following disadvantages:

1. They use structures under the domain of the image activator.

2. Vector pointers are not copied to subprocess by SYS$SPAWN implying that the SYS$SPAWN system service may have to be tampered with to make the facility work for spawned subprocesses.

There is some danger in sharing the same data areas with VMS's image activator, but the new provision for permanent vectors is clearly an intentional feature, though not currently utilized by VMS. Should VMS develop uses for this feature that makes our use of it impossible, we can fall-back on the system-wide vector mechanism which is left entirely to the VAX owner.

Alternately, the vector itself could be revectored to our own code which would be responsible for locating and activating the image activator's tables, leaving the dispatch area solely under the domain of VMS image activation and rundown logic. This is possible since the image activator locates the dispatch area through one pointer, CTL$A_DISPVEC and the VMS code which calls the vectors locates the particular dispatch table through the per-process vectors. (It remains to be strictly verified that the image activator uses only CTL$A_DISPVEC, and not any of the specific vector pointers.)

Since we are using the dispatch area in a fashion consistent with the architecture of VMS, it is unlikely that an unresolvable conflict will occur. However, it is important that contingency mechanisms are available.

3.2 Recovery from Process Deletion

When SYS$DELPRC is called (from whatever process), the PCB of the target process is flagged as delete pending and a kernel mode AST is declared. Process deletion always occurs in the context of the target process. Deletion is accomplished through the AST even if the requesting user is the same as the process targeted for deletion.

Process recovery draws on an analogy between the rundown dispatch mechanism and the search for user-written change mode handlers. If a change mode handler does not recognize a change mode code, it performs and RSB which returns for the search for other handlers. If the handler recognizes the code, it branches to the routine which performs the function. That routine returns with a RET to the last active call-frame which was established by the change mode exception dispatcher.

The environment provided by the kernel mode AST for process deletion coupled with the very early call of the rundown vectors by the SYS$DELPRC system service, makes recovery of the process straight-forward. The rundown interceptor is called by the SYS$DELPRC AST with a JSB instruction. The VMS AST delivery mechanism has called the deletion AST code with a CALLG instruction. Consequently, if the rundown interceptor returns via an RSB instruction, it will return to the deletion AST which will continue through process deletion. If, however, the facility returns via a RET instruction, control is passed back through the last active call frame, i.e., to the AST dispatcher which dismisses the AST, causing the process to continue as though nothing had happened.

Before dismissing the AST to abort process deletion, the delete pending bit in the process's PCB must be cleared. This flag is also useful for verifying that process deletion is the reason that rundown has been requested for kernel mode. It is set by the minimal code in SYS$DELPRC that is executed in the requester's context before queuing the kernel mode AST.

3.3 Forcing System-wide Logout Command File Execution.

Having aborted the deletion of the process, the program must control and force execution of a system-wide logout command file. In other words the intercreptor must cause DCL to execute an "@command_file" type command in the normal process context, but beyond control of the process owner.

The usual way an image causes DCL to execute such a command is to call the run-time library procedure LIB$DO_COMMAND with the desired command as an argument. This procedure is specifically constrained to operate in user mode. The intercreptor runs in kernel mode, and must return to its caller in kernel mode.

The goal is to get to user mode, perform the LIB$DO_COMMAND functions and then restore the environment and return to the caller. The code could force its mode to user, perform its operation, and then return to kernel through a special change-mode-to-kernel dispatcher. An alternate mechanism was selected, however.

The process recovery routine of the interceptor issues a supervisor mode AST just before dismissing its own kernel mode AST. It is the supervisor mode AST that is responsible for altering DCL's course to the logout command procedure.

A supervisor mode AST was selected for the following reasons:

1. By raising mode as soon as possible, any bugchecks incurred are process fatal, rather than system fatal.

2. The convenience of SYS$DCL CMH in altering change-mode dispatching.

3. It is preferable to run in the least privileged (highest) access mode as is feasible.

4. It is "natural" to deal with CLI data structures in supervisor mode, which is where the CLI customarily runs.

Running in the context of the supervisor AST, the I/O on the CLI's input channel is canceled so that there will be no conflict with outstanding read-with-prompts.

Then the AST establishes a user stack context (saving any that is in place). This is done since there is no guarantee that a user mode stack exists. Before access mode is raised to user, a change mode to supervisor handler is declared so that the mode can later be lowered back to supervisor.

The processor mode is then artificially raised to user by fabricating a PC/PSL pair and performing an REI. This provides the unusual situation of a supervisor mode AST running in user mode! A "@command command is sent to the CLI via the callback routine (SYS$CLI) while in user mode.

The reason this user mode operation is considered still part of the supervisor AST is that ASTACT level field in the PCB is still set to supervisor. This level is used as a check to avoid spurious AST's. Consequently, the AST dispatcher still sees an active supervisor AST and protects it.

Note that the supported LIB$DO_COMMAND procedure is not used. Instead the unsupported procedure, SYS$CLI, which is used by LIB$DO_COMMAND is used. This is due to the fact that LIB$DO_COMMAND presumes a "usual" user mode image and performs a call to SYS$EXIT to rundown the image as a final act. There may not be an image to rundown in our situation. If there is not, the execution of SYS$EXIT will cause an "exit pending" flag to be set causing the next user image to exit immediately... an intolerable side-effect.

Since the call to SYS$CLI is modeled after the supported procedure, there is little danger in using the unsupported service. If the CLI callback protocol is changed, it will be changed in LIB$DO_COMMAND and the interceptor's code can be changed accordingly. Effectively, the procedure is supported "one step removed". Future versions may simply use LIB$DO_COMMAND followed by the unconditional clearing of the exit pending bit in the PCB.

The supervisor AST cannot be dismissed while in user mode. Before changing to user mode, a change mode to supervisor handler was declared. We therefore perform a CHMS instruction with the appropriate code. If the code is not that expected, it is passed to any previously declared handler. Otherwise, the previous handler is formally reinstated, the stack is cleared of the CHMS arguments, the user stack is reset to its original context, and the supervisor mode AST is dismissed.

As soon as the supervisor mode AST is dismissed, the system-wide command file is run, as setup by the CLI callback routine.

3.4 Deleting the Process after Command File Execution.

Having executed the system-wide command file, there must be a way of deleting the process which does not cause the command file to be executed again. The prototype described in this paper sets a flag that the CLI call-back has been made for this process. (The flag resides in P1 space with the prototype's code.) If the flag has been set, it means that the command file operation has previously been initiated, and the interceptor returns to process deletion with an RSB instruction.

Note that a user could "pump" consecutive delete process requests against his process from another process. In a production version the command-file flag must be augmented by another which can be set

only from another image by someone with OPER or other suitable privilege. This image would be invoked as the last act of the command procedure. The image can then set the appropriate privilege protected flag, based on the history of the process (in kernel mode we can get any privilege we want). Only when this flag has been set will the process resume deletion. If the command-file flag is set, the deletion AST will be dismissed, but no attempt will be made to initiate the logout command procedure.

An operator's utility should also be developed to force deletion of a process, circumventing or aborting the logout procedures.

3.5 Image Residency

It is important that the rundown intereceptor be in a permanent segment of memory which will survive across image activation and rundown. This eliminates P0 space since it is destroyed entirely at each image rundown.

The two candidates are P1 space and S0 space.

Image rundown causes all P1 space beyond the address contained at CTL$GL_CTLBASVA to be deleted. Therefore, if the image is loaded by extending P1 space, then the value at CTL$GL_CTLBASVA must be changed to point to the new end of P1 space. This makes the code permanently mapped to the process's P1 space.

Loading S0 space is simply a matter of allocating contiguous pages of paged or non-paged pool and loading the image. S0 space MUST be used if the system-wide VMS rundown intereception vector is used.

3.6 Image Loading

The code to be executed is placed in P1 space by a P0 based image (/P0IMAGE qualifier used in conjunction with the link of the mapping image).

The create and map section system service, SYS$CRMPSC can be used to map any file into memory. It can be mapped as a shareable or a private copy. This system service is used repeatedly by the image activator, SYS$IMGACT to map image sections piecemeal. The easiest way to use this service is to load an image linked "/SYSTEM", i.e., there is no image header.

Since the intereceptor must be strictly position-independent, it can not use general-mode addressing, or external run-time images. If any descriptors are used, they must be initialized at run-time.

You may also use SYS$IMGACT to load the rundown interpreter. The image activator does not perform merged activations directly into P1, so a technique used by VMS to map the CLI into P1 space is employed. The image activator is invoked to merge the image into P0 space for sizing purposes. Having determined the size of the image, the P0 space is deleted and P1 is expanded the appropriate number of pages. The image activator is called again to explicitly map into the expanded address range in P1.

The advantages of SYS$CRMPSC are:

1. It is a fully supported system service.

The disadvantages of SYS$CRMPSC are:

1. The rundown intereceptor must be strictly position independent.
2. External images can not be activated or accessed at run-time.

The advantages of SYS$IMGACT are:

1. Non-PIC code is convertible to PIC by SYS$IMGACT so descriptors and general mode addressing can be used freely. (This conversion is done via a call to a sister system service, SYS$IMGFIX which is considered part of the image activator, though called separately.)
2. Run-time libraries may be used (cautiously).
3. Page protection is automatically handled.

The disadvantages of SYS$IMGACT are:

1. It is not a supported system service. However, it is so ingrained in the basic architecture of VMS, that changes in functionality causing incompatibility with this application are highly unlikely. Changes in calling protocol are more likely.

An early version of the prototype used SYS$CRMPSC system service. The PIC code requirement was unacceptably constraining.

The image activator is so powerful a tool, that its use is warranted in this particular instance, despite its lack of formal support.

To isolate image activator changes, a family of utility subroutines should be developed. Should the image activator protocol change, only this family of routines will have to be changed for maintenance.

3.7 LOGINOUT Inadequacy

VMS's LOGINOUT image will not be usable for logout since it closes the process permanent files (SYS$OUTPUT, etc.) before calling SYS$EXIT from kernel mode (which eventually calls SYS$DELPRC or SYS$RUNDWN). Under this circumstance, the intereceptor still gets control, but has no communication channels for running a logout command file.

Command files typically make use of these files. It is unreasonable to expect logout components to make no use of these standard files. Therefore logouts must be effected by an image which calls SYS$DELPRC and leaves the files intact. This will provide a wholly normal process context in which the system-wide logout command file may run.

It is natural to ask, "What is given up?". The LOGINOUT image makes calls to security audit software, and does a couple of other minor operations.

The important thing is that these operations are all intrinsically optional. The LOGINOUT image can be bypassed very simply by an UNPRIVILEGED user by:

1. Issuing the DCL command "$ STOP/ID=0", or
2. Calling SYS$DELPRC from an image.

On the other hand, logout operations implemented through the proposed rundown intereceptor cannot be bypassed by an unprivileged user.

4 Prototype Rundown Intereceptor

A prototype rundown interceptor and loader has been successfully implemented using techniques described in the section on problem discussion and solutions.

!!! WARNING !!!

The prototype code supplied on the DECUS tape and described herein is strictly experimental and is supplied for demonstration purposes only. The code has known deficiencies and is not intended for production purposes. The code is intended to be used by system programmers as a "template" for studying the problem.

The per-process vectors were selected for the prototype due to the accessibility for development purposes, and so that the code could be tested with others on the system. (Once a system-wide vector is initialized it is used for every process on the system). The use of the per-process vectors also provided a familiarity with the vector usage protocol which can be used as a model of a protocol for system-wide vector usage, if desired.

P1 space was selected for the prototype for the following reasons:

1. Ease of access.
2. Minimal perturbation to system during test.

The loader uses the image activator to load the interceptor into P1 space and sets up the per-process rundown dispatch vector to activate the rundown interceptor.

The interceptor assumes that logout is performed by a SYS$DELPRC system service call, rather than running the LOGINOUT image. (The DCL command "STOP/ID=0" equivalent). An undeletable process will result if normal logout is attempted.

Note that the interceptor in the prototype is activated from the image activator's dispatch area. When it dismisses the deletion AST, none of the remaining vectors are activated. This is not tolerable for a production version. If P1 dispatching is retained then the remaining vectors can be activated by a JSB indirect on the stack pointer, or they can be run and eliminated by a call to SYS$RUNDWN for user mode. If S0 dispatching is employed, the P1 vectors have all been called and completed by the time the interceptor is activated.

The prototype uses an overly simplistic mechanism for resuming process deletion after command file execution. See the section on resumption of deletion for details.

The prototype interceptor outputs messages showing significant events in its execution for demonstration purposes.

The interceptor works for:

1. A top-level process.
2. A spawned subprocess.
3. A top-level process which had deletion requested by any other process.

Please note that the message which marks resumption of supervisor mode will have to be moved to make the third scenario operate.

Thoroughly commented source and build files are supplied on the DECUS tape for this Symposium. Details on the implementation algorithms are included as well as procedures for building a "demo" model.

5 Path to a Production Version.

5.1 Program Residency

A number of factors point to the desirability of using S0 space rather than P1 space for the image.

The primary factor is that the P1 vectors are a part of the image activator's data area. Prudence suggests that we stay away from this area if there is another reasonable approach to accomplish the same thing. The use of P1 residency would require the use of the P1 vectors.

Another factor is that the SYS$SPAWN system service does not copy extensions to the parent process's address space into its own. The SYS$SPAWN service starts with a new shell, initializes it, and then, through a series of mailbox communications between the parent and spawned process, the logical name tables and CLI symbol tables are optionally copied. If the rundown interceptor were to be mapped in S0 space, no such problem exists.

A routine for mapping an image into S0 paged or non-paged pool must be developed. No tests have been made of loading S0 space with the image activator; however, it is anticipated that no major difficulty will be encountered in its use.

5.2 Vector Mechanism

This issue is related to the image residency problem. If the image is to reside in P1 space then the per-process vector must be used, since uniform mapping of P1 space across processes can NOT be presumed. On the other hand if S0 space is selected, then either vector can be used. However, per-process vectors are not copied to a spawned subprocess.

Use of the S0 vector is currently indicated. It is desirable to avoid sharing the image activator's data area. Per-process capabilities can be provided by keeping a list of per-process intercception vectors as in the image activators per-image vectors. A set of utilities can be developed to manage these data areas.

5.3 Environmental Flexibility

Process deletion is invokable in a number of fashions, only a few of which have been tested. For example, the situation of the deletion of a disconnected process which has timed out must be investigated. Tests on the availability of input/output channels must be made and appropriate contingency action developed.

An inventory of process deletion scenarios must be made and appropriate special action (if any) developed for each case.

Some of the scenarios will require special considerations in order to accommodate them. For example, if a process is being deleted because of CPU limit time-out, one or more optional procedures can apply. On first encounter, one may artificially boost the time limit enough to initiate and finish logout procedures. On second encounter, we know that our estimate on the time for logout operations is wrong, or that there is something seriously wrong with the logout code or the machine itself -- an errorlog entry and continuation through deletion is in order.

5.4 Error Mode Analysis

The system must be revisited to implement error-mode detection and contingency operations (particularly in considering the various scenarios of process deletion as discussed above). In general, any error mode must proceed toward normal process deletion.

5.5 Program Structure

If the image activator is used, it will be possible to link at run-time to base functionality, allowing independent maintenance of functional subsystems, e.g., process rundown and image rundown functions.

5.6 System Programming Guidelines

The implementation of the rundown interceptor uses some of VMS's one-of-a-kind hooks. System programmers may need to use the VMS facilities used by this application. In order to do this a comprehensive set of system programming conventions and utilities must be developed to allow system programmers to achieve the effect of the VMS hooks used by this application. In fact, the resulting set of tools could make the use of those VMS facilities easier than without the application.

Protocols and conventions must be developed to coordinate the use of:

1. P1 and S0 Space
2. P1 and S0 Vectors
3. Per-process data conventions and utilities
4. Processor Register Conventions

5.7 Logout driver to replace LOGINOUT

A logout driver must be developed to emulate the user-interface of LOGINOUT. However, the new driver will not close process permanent I/O channels. Extended features can be developed, e.g., clear the screen as a final act leaving no print on the screen whatsoever.

The new driver will terminate the process via a call to SYS$DELPRC.

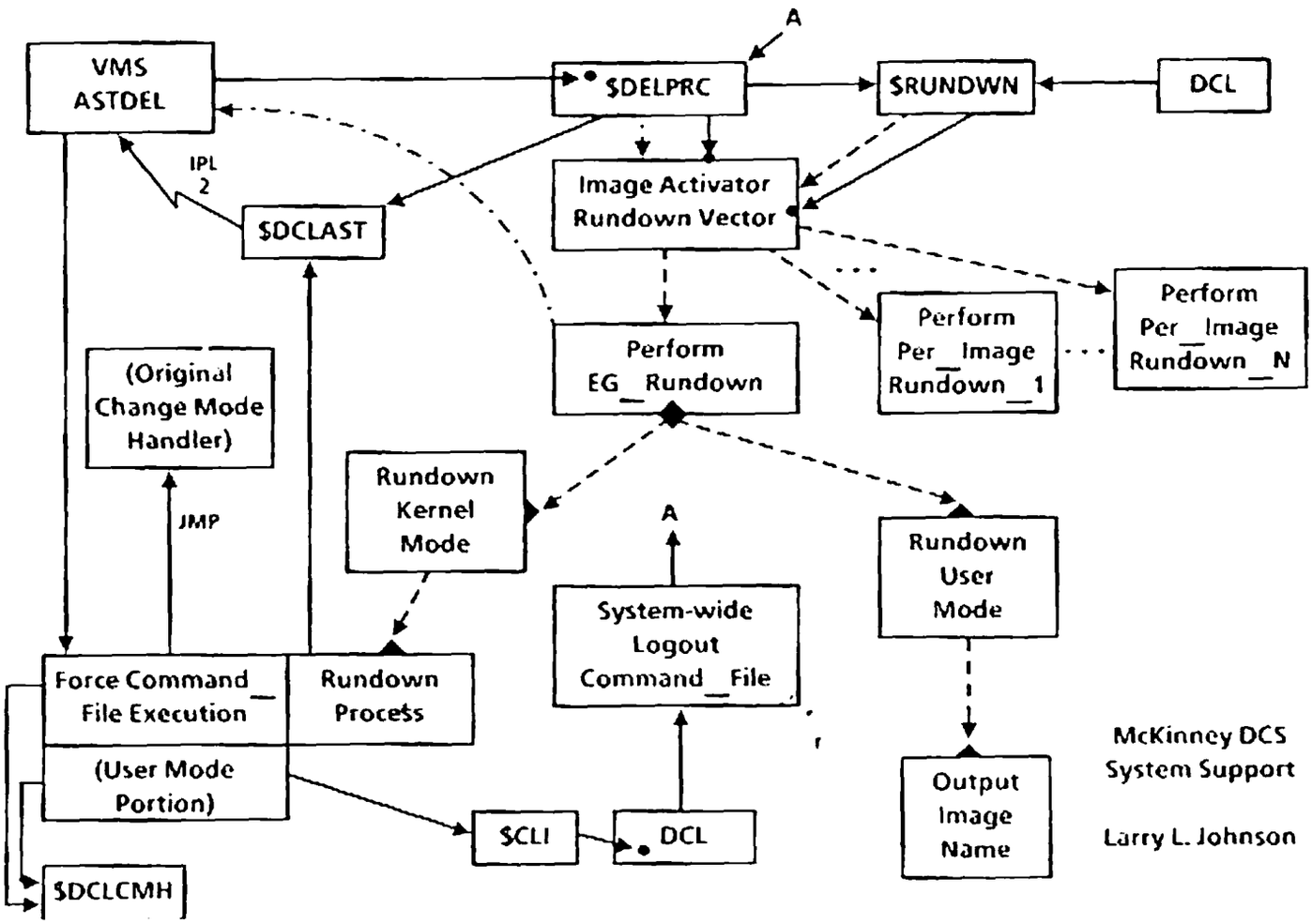
Investigations into running LOGINOUT at the end of the command procedure will be made. The image could be protected from direct operation through ACL's specially constructed to allow access during login and post-logout-file operation only.

APPENDIX A

STRUCTURE OF RUNDOWN INTERCEPTOR

The following structure chart outlines the call structure of the rundown interceptor.

Rundown Interceptor Prototype Call Structure
November 1985



```
**********************************
***          Notes to          ***
***     Rundown Interceptor    ***
***       Structure Chart      ***
**********************************
```

Module descriptions are found in the code listings. Modules formatted "$name" are standard VMS system services SYS$name.

The SYS$RUNDWN system service is not supported. It is shown here to describe its role.

The VMS_ASTDEL module is actually part of the VMS executive, the Asynchronous System Trap Delivery Mechanism. It is shown here to describe its role.

The SYS$CLI service is unsupported. Its use is modeled after the LIB$DO COMMAND procedure of the VMS run-time library.

The DCL module shows the CLI's role in the command file activation. It represents the command interpreter.

```
*******************************
*** Structure Chart Legend ***
*******************************
```

A. Solid lines indicate activation via VMS Procedure Calling Standard (CALLx/RET), except for the activation of the command procedure by DCL which is the usual "@command-file" activation.

B. Dashed lines indicate activation by a JSB protocol.

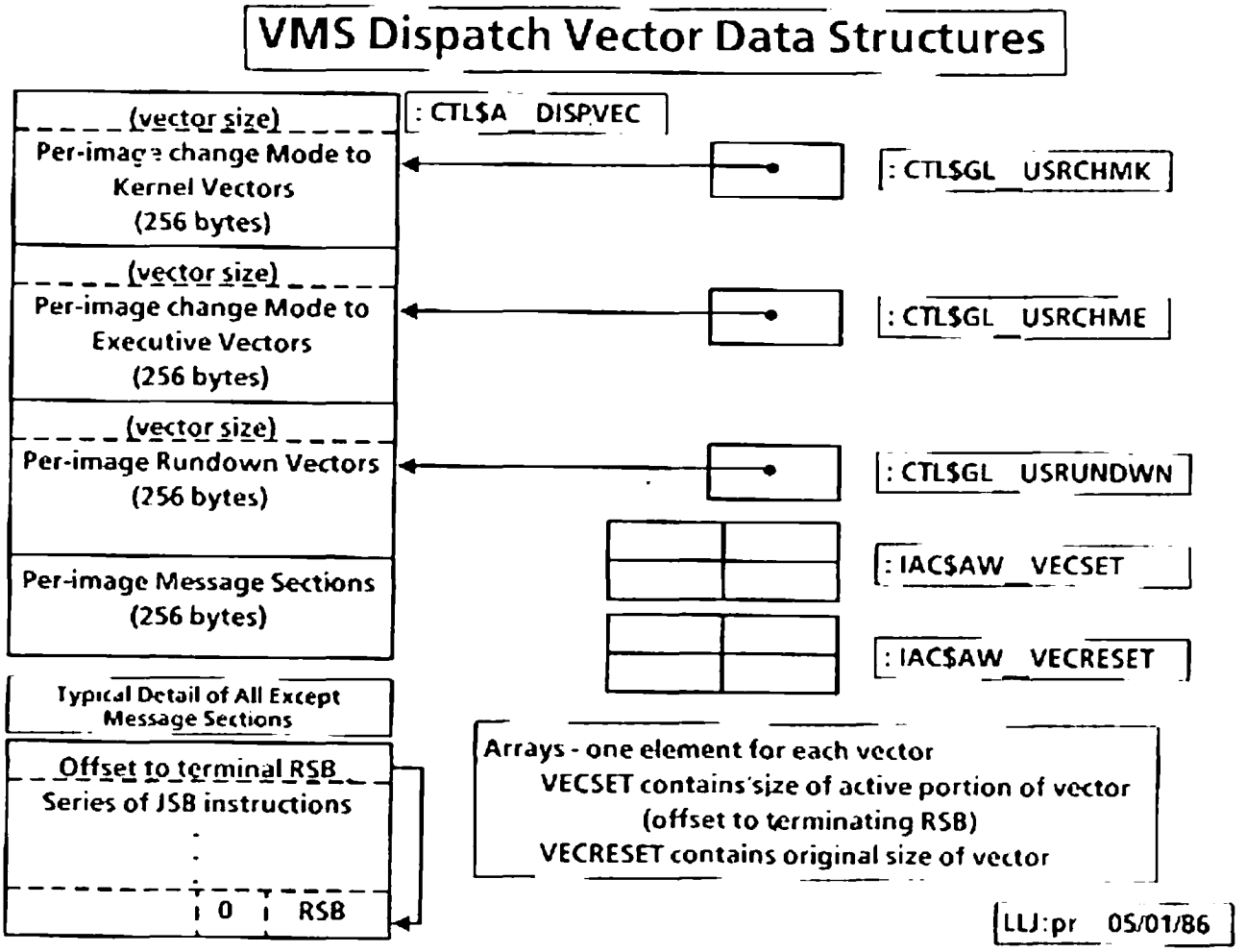
C. The dash-dot line indicates the issue of the RET to the last active call frame by a procedure which was actually called under the JSB protocol.

D. Triangular "hats" on modules indicate lexical inclusion within the subordinate, i.e., they are in the same compile module.

E. $DELPRC invocation (A) is the last act of system-wide logout command procedure.

APPENDIX B

LAYOUT OF VMS P1 DISPATCH VECTOR AREA



APPENDIX C

LISTINGS OF RUNDOWN INTERCEPTOR

APPENDIX D

LISTINGS OF RUNDOWN INTERCEPTOR LOADER

APPENDIX E

LISTINGS OF MESSAGE FILES