An Exposition on SAV in RSX-11M-PLUS V2.0

Paul J. Bezeredi

Digital Equipment Corporation
Nashua, New Hampshire

I. Why SAV?

The general intent of SAV is to copy the current system image from main memory back into the system image disk file from which the system was booted. This causes the current state of the running system, including modifications made since the system was booted, to become a permanent part of the system image file. Therefore the modifications do not have to be made whenever the system is booted. For example, tasks do not have to be reinstalled every time the system is booted and patches (made by OPEN or XDT) need be made only once.

SAV, and the indirect command file it invokes, will automatically handle the many details necessary to use any system. For example, redirecting the pseudo devices, mounting the system disk, and utilizing all of available main memory.

There are two other reasons for saving a system. SAV provides the only way a system image can be made hardware bootable, that is, for the disk's hardware boot block, Logical Block Number (LBN) 0, to be written so it points to the system image. Additionally, after being saved, the disk on which the system resides can be compressed (copied by BRU or DSC) without destroying the integrity of the system.

II. The Concepts and Techniques of SAV

SAV operates in two distinct phases. In the first phase, SAV writes what is in main memory into the system image file, and it is executed only by invoking SAV from MCR. In the second phase, SAV restores the saved context to the CPU and it is invoked either automatically in the first phase or by booting the saved system image file.

In phase one, the lowest section of main memory (bytes 0 through LOWSIZ*2) is copied into a buffer in SAV's address space (see the module SAVE for the definition of LOWSIZ). The system image bootstrap (see $VBN1 in the module SAVVB1) is copied into the locations just vacated. The special driver for the type of system disk is copied into the system image bootstrap. (See Section V, "Special Drivers", for details about special drivers.) The mapping registers are pushed onto SAV's stack and a copy of the the special driver that is in SAV's address space writes all of main memory into the system image file.

Basically, phase two reverses that process. Control of the CPU is passed to the system image bootstrap from phase one by jumping to location zero or by using BOO, which reads the system image bootstrap from the front of the system image file into low main memory.

The special driver in the system image bootstrap reads the rest of the system image file into main memory. Control is passed to the task SAV, which overwrites the system image bootstrap with the orginal contents of low main memory, restores the mapping registers from the stack, performs housekeeping functions (for example, expanding or contracting the last partition in the system to include all of available main memory, redirecting the pseudo devices SY:, LB:, and SP: to the booted device, and initiating the startup indirect command file), and exits.

A part of phase one and two deserves special mention. Every installed task has a Task Control Block (TCB) in the system data structures. Offsets T.LBN and T.LBN+1 contain the LBN of the task image file on the disk, so the Executive can quickly locate the file for initial task loading. If the disk is compressed (copied) by BRU or DSC, the task images will probably change locations on the disk, and therefore the system data structures will be wrong. To avoid this, SAV retrieves the task's File ID, which does not change when a disk is compressed, from the task's header and saves the File ID in the task's TCB when the system is written out. SAV converts the File IDs back into LBNs when the system image file is read into main memory.

All active copies of multiuser tasks have a similar problem. Their Partition Control Blocks (PCBs) contain the LBNs of their common read-only sections. When the system image is written out, SAV converts those LBNs to numbers relative to the first block of the task image files that contain the common read-only sections. SAV reconverts the relative block numbers back into absolute block numbers when the system image file is read into main memory.

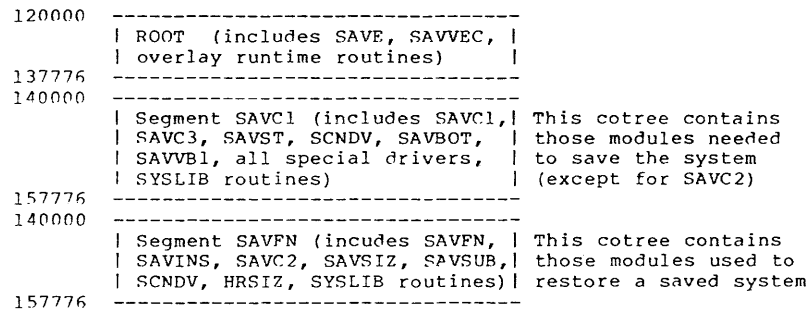
The M-PLUS version of SAV is memory resident overlaid for two reasons. First there is more code in SAV to fit into the 8K words allotted to a privileged task that references the I/O page. The modules SAVC1, SAVC3, SAVC2, SAVST, and SAVFN all used to be part of the module SAVE. There were no subtle reasons for breaking SAVE up into those particular modules. The present arrangement was chosen because it made both memory resident overlays about the same size. Second, because each of

the memory resident overlays is loaded before main memory is copied to the system image file, SAV.TSK can be deleted after the system is saved, and the system will still successfully boot, as long as SAV does not have to be checkpointed in order to mount the system disk.

In the future M-PLUS may support new system disks. Therefore SAV will have to add more special drivers. Such an upgrade should not require reoverlaying SAV (should not increase its virtual address space) but should require modifying the fewest possible modules. Adding another special driver merely requires the writing of a new special driver module and the updating of SAVBLD.ODL.

Because manual overlay loads cannot be mixed with automatic overlay loads, the loading of memory resident code segments such as SAVC1 is also done manually. This is a bit unfortunate, because the implicit loading of code segments is both less confusing and less cluttered. Figure 1 gives a graphic description of the memory layout of SAV.

FIGURE 1 -- MODULE LAYOUT OF SAV.TSK



In M-PLUS, SAV, BOO, and VMR all share a technique for determining the length of a system image bootstrap. SAV symbolically defines the number of disk blocks that compose the bootstrap and derives the values of BTADD and LOWSIZ from this. BOO and VMR get the length of the system image bootstrap minus one from the "transfer address" of the system image file's label block, where SAV leaves it. As a consequence, M-PLUS can boot RSX-11M systems, which have a smaller system image bootstrap, but by coincidence (as far as RSX-11M is concerned) have the appropriate value (zero) as the "transfer address."

In RSX-11M, SAV, BOO, and VMR all locally define the length of a system image bootstrap, BTADD, and LOWSIZ (watch out, all the references may not even be done symbolically). As a consequence, RSX-11M pre-V4.0 systems cannot boot M-PLUS systems, because the 11M BOO has no way of knowing that it must read in more than the one block that is normal for a saved RSX-11M system image file.

The current state of RSX-11M V4.0 is that BOO has been modified to interpret the "transfer address" of a system image file in the same manner as M-PLUS. That is, BOO can boot both RSX-11M and M-PLUS systems. SAV needs to be modified to consolidate the definitions of BTADD and LOWSIZ and to always use the symbolic definitions. VMR (see the modules RANIO and SETUP) needs to be modified to use the "transfer address" from the system image file's label block rather than explicitly knowing how big the system image file bootstrap is.

III. SAV's Restrictions

The restrictions imposed by SAV come in two general categories: those detailed checks on the state of a system that must be passed to guarantee that the system can be successfully saved and the general implications of the current structure of SAV.

For a system to be saved, it must be in a specific state, that is defined in general terms as being "quiet" or inactive. Checks that are made are sensitive to the detailed capabilities of the system (see Section IX, "What SAV does in Detail") and therefore change from base level to base level. When a check fails, you will receive a specific error message.

The manner in which SAV is currently implemented results in several general restrictions:

- If a system is saved on one type of disk (for example an RL02), it generally cannot be copied by BRU or DSC to another type of disk (for example an RK06) and remain bootable. This is because the driver in the system image bootstrap can generally boot only one generic type of device (in the example, this would be an RL01 or RL02). The exception to this are MASSBUS devices (RM02/03/05/RM80, ML11, RP04/05/06/07), RK611/711 devices (RK06 and RK07), and UDA50 devices (RA80). A significant decrease in packaging complexity has been achieved for M-PLUS by combining the MASSBUS, RK611, and UDA50 special drivers into one common driver (in the module SAVCM) and using a controller detection algorithm and the device drivetype code supplied by the hardware to determine the disk type and its associated geometry.

- A saved system cannot be booted if the hardware boot block or the system image boot blocks contain ECC correctable (or any other) errors. This is because neither the ROM boot code, which reads in the hardware bootstrap block, nor the special driver in the hardware bootstrap block, which reads in the system image bootstrap, is large enough to perform ECC correction. They merely retry an unsuccessful operation, which causes an infinite loop, or simply halt.

- Because the M-PLUS system image file bootstrap is more than one block long, its special driver is big enough to handle ECC correctable errors in the system image file. Because the system image file bootstrap for RSX-11M is only one block long, it is too small to hold the ECC correction code. Therefore the RSX-11M boot process is vulnerable to an ECC correctable error anywhere in the system image file.

While an M-PLUS system can boot an RSX-11M system, the inverse is not necessarily true. There are two theoretical reasons for this. First, a saved M-PLUS system requires that CPU registers R4 and R5 contain the physical unit number and CSR address of the boot device. A pre-V4.0 RSX-11M system does not pass this information. (For a complete description of register conventions, see the module SAVVB1.) Second, when reading a system image bootstrap into main memory, a pre-V4.0 RSX-11M system reads in only one block. An M-PLUS system image bootstrap consists of multiple disk blocks.

The attempt to boot a saved M-PLUS system image file with an RSX-11M pre-V4.0 BOO will result in a NO TRANSFER ADDRESS error message before either of the two problems are encountered. This is because the "transfer address" of the saved M-PLUS system is currently 1. The pre-V4.0 BOO mistakes this for the transfer address of a virgin system image file and rejects it because it is odd.

Note the implications of the /CSR switch (see Section VI, "Format of the SAV Command Line"). If the switch is not used, a system can be saved and booted on any controller on the system as long as the hardware bootstrap returns the CSR address of the controller in CPU register R1. To the best of our knowledge, all bootstraps that M-PLUS is likely to run on behave in this fashion.

Debugging SAV is tedious. It is a privileged task that must reference the I/O page, so it is restricted to a maximum size of 8K words. In spite of being overlaid, SAV is too big to allow the inclusion of ODT. The trick of using ZAP to insert BPT instructions in sections of code that run in Kernel Mode (and using XDT as the debugging aid) doesn't always work, because SAV frequently gets into Kernel Mode by jamming the Program Status Word rather than using CALL $SWSTK. The result of this practice is that $STKDP is greater than 0 when the Executive handles the breakpoint and therefore the Executive does not correctly determine how to handle the trap.

There is an additional complication. When exiting XDT, the various Page Descriptor Registers (PDRs) are jammed to be appropriately read/write or read-only. They must all be read/write for parts of SAV. See the module SAVE to identify which sections are restricted in this manner.

The result of this is that infinite loops inserted by ZAP and manipulated from a debugging console are frequently the debugging method of choice.

Many parts of SAV (for example, the memory sizing routine in the module SAVSIZ), manipulate the memory management registers. They obviously must not remap the register that maps their instructions. As a result, such sections of code must be mapped by certain Active Page Registers (APRs). SAVBLD.ODL identifies such sections and the APRs that must be used to map them.

IV. Virgin Systems

The task builder produces virgin (unsaved) system image files. They are characterized by having transfer addresses in their label blocks that point to INITL or XDT. Various fields in the system's data structures, such as T.LBN in each TCB, contain Logical Block Numbers. A saved system image file has a "transfer address" of less than 60 (octal), SAV as the active task, and such words as T.LBN in each TCB containing pointers that are invariant across disk compressions.

Although useful work can be accomplished using a virgin system by manually mounting the system disk, establishing a checkpoint file, bringing devices online, etc., this is not a design goal of M-PLUS. The system may be limited in the future so that the only legitimate operation that can be performed using a virgin system is saving.

A virgin system image will not generally function correctly if its disk has been compressed (copied by BRU or DSC). This is because the Task Control Blocks (TCBs) of the installed tasks and some Partition Control Blocks (PCBs) of checkpointable commons contain absolute pointers to disk LBNs. When a disk is compressed, the starting LBNs of common and task image files generally change. As explained previously, one of the big tricks of SAV is to convert those absolute pointers into pointers that are invariant across disk compressions.

A virgin system image file can be copied by using PIP. If a virgin system image file is on a disk that has been compressed, it can be salvaged by using VMR to remove all tasks and commons and then reinstalling them.

Because a virgin system does not size the system disk, SYSGEN must produce a Unit Control Block (UCB) for the system disk that correctly indicates its geometry (for example, an RK06 or RK07) if the system is to work once it has been booted.

Because of a "trick" that can be used when answering the SYSGEN peripheral questions, there is a bit more flexibility available than is immediately obvious. Currently, for the RL01/02, RP04/05/06, and the RM02/03, the only difference between the large disk of the generic type (an RP06) and its equivalent small disk (an RP04) is the number of cylinders per disk. Therefore when you are given a choice of drive types, choose the largest type. However, note that the RK06/07, RM80/RM05, and RP07 must be dealt with honestly. If the drive turns out to be the larger type, everything is perfect. If the drive turns out to be the smaller, it cannot have LBNs that are too big (LBNs that would push the heads off the end of the disk).

A virgin system does not have to be "quieted" by a CON OFFLINE ALL command before being saved. The I/O data structures output by SYSGEN show all the appropriate controllers and units (everything except the booted disk and console) as offline.

V. Special Drivers

Special drivers are used by SAV to write the contents of memory to the system image file and to read the system image file into main memory when the saved system image file is booted. When SAV writes the hardware boot block, SAV includes a special driver that will read in the system image bootstrap of the saved system image file. The source modules for SAV's special drivers are found in UFD [12,10] and are named SAVdd, where dd is the device name. For example, SAVDL is for the RL01/02 and SAVCM is for the MASSBUS, RK06/07, and UDA50 devices.

BOO also uses special drivers to read system image files into memory. BOO can handle DECtape, the RX01, and the RX02 (see the module BOODRV) in addition to the devices SAV can handle.

A special driver is special because of the constraints under which it must work. A special driver must not use interrupt vectors (while the special driver is being used, the system image bootstrap occupies low main memory). It must be as small as possible (because it must fit into the hardware boot block). It must conform to an elaborate set of conventions established because the driver must be able to read or write variable amounts of main memory or system image files.

The small size (you do not want to waste instructions making tests) and need for flexibility are met by using SAV to modify the driver before inserting it into one of the various environments (hardware boot block, system image file bootstrap, or internally in SAV) the special driver can handle. The places to make the modifications (for example, where to start writing, whether to write or read, etc.) are specified for each special driver by a table.

Structurally, a special driver consists of a module that has three sections:

- o The first defines the names of the CSR offsets that the driver uses.
- o The second is a fixed length segment of the PSECT DRVTAB that defines various aspects of the driver. For example, its length, what a write function is, and what it's segment name is. In the aggregate, the segments form a contiguous table that is terminated by a zero word.
- o The third section is the driver itself.

When the driver is entered, the following CPU registers have special significance:

R5 - will be zero (indicating that the driver should use the CSR address stored in the driver) or nonzero (indicating that the contents of R5 should be used as the CSR address).

R2 - will be zero if no UMRs are required by the special driver else it will be nonzero.

R3 - will contain the BAE offset if R2=0 and the special driver being used is SAVCM or SAVDL else it will be undefined.

Locations 4 and 6 will contain the LBN where the driver should start reading or writing.

When the driver has finished, the following registers will contain information required by SAV:

R0 - will be the residual block count (the number of blocks that were to have been read or written, but were not).

R1 - will be the ASCII representation of the load device name, that is the name of the device that was just used.

R4 - will be the physical unit number of the device that was just used.

R5 - will be the CSR address of the device that was just used.

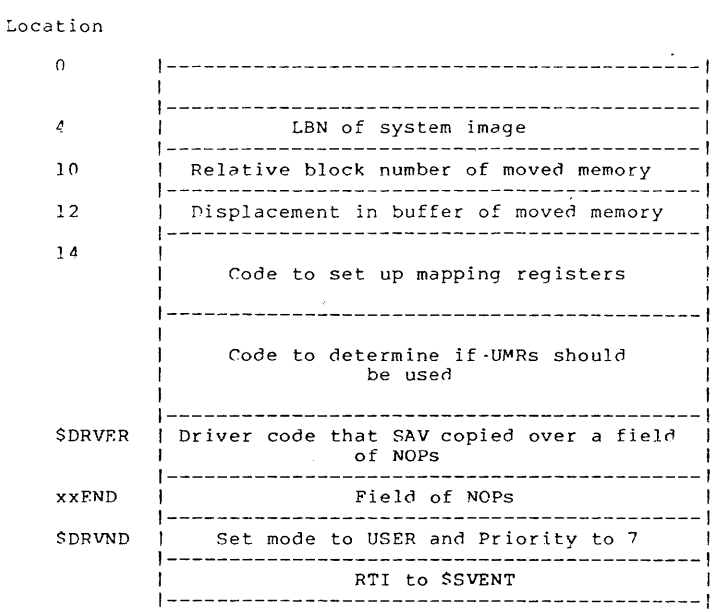
Although the best way to write a special driver is to follow closely one of the existing special drivers, several subtleties deserve mention.

All drivers end with a RETURN (labelled xxEND, where xx is the name of the special driver). When SAV copies the special driver into either the hardware boot block or the system image file bootstrap, it copies up to, but does not include, the xxEND statement. Thus when the driver is finished, control falls into the next instruction of the bootstrap. When SAV uses the special driver to write main memory to the system image file, the special driver is called, so the xxEND statement returns control to SAV. Figure 2 gives a graphic description of how the special driver fits into the system image bootstrap.

Special drivers must be extremely careful about using the stack, because there is not much available. For example, a special driver is about to read in the system image bootstrap into locations 0-2776. The stack is at 3004. If the driver pushes three words onto the stack before starting the transfer, the last word will be overwritten by the system image bootstrap.

Additionally, the use of a "trap catcher" by the special driver can easily push enough onto the stack to extend it into the last few words of the boot block.

FIGURE 2 -- SYSTEM IMAGE BOOTSTRAP (VBN1-3)



VI. Format of the SAV Command Line

The format of the SAV command line is:

SAV[E] [/WB][/MOU="switches"][/SFILE="file spec"][CSR=x]

/WB is an optional switch that indicates that the hardware boot block (LBN 0) of the boot device should be rewritten to point to the system image file that is about to be written.

/MOU="switches" is an optional switch that provides a string to append to the MOUNT command for the system disk. This allows the overriding of the default mount parameters for the system disk. The pair of double quotes is part of the required syntax. SAV simply appends all the characters between the quotes to the MOUNT command. It does no syntax checking.

/SFILE="filespec" is an optional switch that provides the file specification of an MCR indirect command file to use in place of "booted-device:[1,2]STARTUP.CMD", which is the default file specification. The pair of double quotes is part of the required syntax. SAV does no syntax checking on the file specification. It simply precedes the characters between the quotes with an "@" and queues the resulting string to MCR.

/CSR=x is an optional switch that specifies the CSR address for the boot device. If this keyword is not specified, SAV uses the CSR address in the boot device's KRB for writing the system image out to the disk and what the hardware bootstrap leaves in CPU register R1 for reading it in when the system is booted. If x is an even octal number greater than 157777, SAV uses the KRB CSR address to write the system out and x to read it in. If x is "SY", SAV uses the KRB CSR address to both write the system out and to read it in. An implication of this is that given the proper type of hardware bootstrap, the disk can be booted on any controller in the system, not just on the controller on which the system was saved.

This switch is useful for creating distribution kits on systems where the CSR of the actual device is not at the default CSR address for that type of device. For example, on a system with RP05s and RM03s, the CSR address for the RM03 will not be the default, because the RP05s occupy the default address. To create a disk that will boot on a default RM03, save the system with CSR=176700. The system will reboot (the CSR address used to write the system out is taken from the RM03's KRB, and the system image bootstrap that has been left in low main memory is forced to use that CSR address to read the system back in). Use the CON command in VMR to set the vector and CSR addresses for the RM03 to the defaults.

VII. A Roadmap of SAV's Modules and Entry Points

Module	Entry Point	Function
SAVE	SAVEP	Load overlays and call subroutines, push mapping registers onto stack, copy low main memory into a buffer, put the system image bootstrap in low main memory, and transfer control to it
	$SVENT	Restore low main memory, hardware map registers, and CPU context
	$KRBOF	Put KRB offline and remove vector
SAVC1	$SAVC1	Parse command line, unstop tasks, tell PMT to exit
SAVC3	$SAVC3	Check memory size, checkpoint all R/W commons, insure checkpoint files are inactive, check active processors
SAVC2	$SAVC2	Dismount load device, check for tasks with outstanding I/O, insure all devices are offline
SAVST	$SAVST	Set up special driver, check home block, and write label block

	$SAVDN	Setup hardware boot block, set TI: and SY: offline
SAVINS	$STFID	Convert T.LBNs and P.LBNs
	$INSTK	Restore the T.LBNs and P.LBNs
SAVVB1	$VBN1	Initialize the CPU, read system image file into main memory, and transfer control to it
SAVSIZ	$STCLK	Select clock and initialize clock and Floating Point Processor vectors
	$TSTPY	Set up nonexistent CSR table
	$CRSIZ	Size main memory
	$STCPU	Determine CPU type
HRSIZ	$SDISK	Size the booted device
SAVFN	$SAVFN	Finish bringing the system up
	$SAVID	Output console ID, redirect pseudo devices

VIII. What SAV does in Detail (Saving the System)

The following is basically a nondefinitive, English version of SAV's code (see modules SAVE, SAVC1, SAVC3, SAVC2, SAVST, SAVFN, SAVVB1, SAVSIZ, SAVINS, and SAVSUB). It is intended to give a flavor of that code and some of its motivation. The code, not this, is the truth.

The principal reason for the order of the following tests is for coding convenience or for historically obscured eccentricities of the various authors. For example, there isn't a logical reason for testing whether or not SAV supports the device on which the system image file is to be written before checking the syntactic validity of the command. At one point in time, it saved a few words of code.

1. If the SAV command was not issued from a privileged terminal, SAV issues the PRIVILEGED COMMAND error message and exits.

2. The name and logical unit number of the system disk (frequently referred to as the load or boot device) is retrieved from $SYSIZ+2 and $SYSIZ+6. They were left there by BOO, which had to know that information to find the system image file to boot.

   The device name is checked against the names of all the I/O devices in the system. If no match is found, the NOT VALID SAVE DEVICE error message is issued and SAV exits. If this occurs, the locations around $SYSIZ have probably been corrupted. INITL would not have allowed the system to come up if the boot device was not in the system I/O data structures.

3. The SAV command line is obtained. If there is an error, SAV outputs the COMMAND I/O ERROR message and exits.

4. SAV sets bit FE.MXT in the first system feature mask word ($FMASK) and unstops tasks that are not stopped for an event flag or buffered I/O. This should cause all Command Line Interpreters (CLIs) to exit because they should check for FE.MXT being asserted before they stop themselves. As a side effect, other tasks, such as HRC..., will also exit. This makes such tasks inactive when the system is copied into the system image file. Thus VMR can remove and reinstall new copies of those tasks.

   Note that SAV will hang if any CLI does not exit. This is only the first example of a general principle. There are systems that cannot be saved. There is an implicit pact between the system as a whole and SAV to provide a "reasonable" environment. There are some cases SAV cannot win, so why try unreasonably hard?

   This philosophy (justified in part because SAV does not have enough address space to be really paranoiac) is rarely challenged simply because SAV is almost solely used immediately after booting a virgin system image file.

5. SAV makes sure that the pool monitor task (PMT) is inactive. If it is installed, an attempt is made to force it to exit.

6. The command line is parsed. If it is syntactically incorrect, a SYNTAX ERROR message is output, and SAV exits.

7. SAV makes sure that the system image file is big enough to hold everything in main memory that is of value (defined to be secondary pool and everything that has a PCB except device commons). If that is not the case, a COMMON, DRIVER OR TASK ABOVE SYSTEM IMAGE FILE LIMIT error message is printed, and SAV exits.

   Either make the system image file large enough or use the MCR PAR command to determine what is above the system image file size and get rid of it or move it lower.

8. SAV attempts to force installed, resident, read/write commons into their file images. It is assumed that a common will remain checkpointed once it is checkpointed, because there is supposed to be minimal system activity when a system is saved. If this is not true, SAV will hang.

It is not obvious why this step is not done before checking to see if all of main memory will fit into the system image file. If all nonreferenced commons were forced out of main memory, the previous check would succeed more frequently.

9. If any checkpoint files are in use, a CHECKPOINT FILE STILL IN USE ON ddn: error message is output for the first such file, and SAV exits.

   ACS should be used to disable checkpointing on that device.

10. If error logging is still active, an ERROR LOGGING STILL ACTIVE error message is output, and SAV exits.

    Run ELI to stop error logging.

11. If there are any checkpointable commons that are not installed from an LB:, a COMMON name NOT INSTALLED FROM AN LB: error message is output for the first such common, and SAV exits.

    When a saved system image is booted, the task image files of all installed commons and tasks are checked to see if they are still usable (for example, that they have not been deleted or that the disk they reside on is still accessible.) Tasks and commons that have unusable task image files are removed from the system. An obvious way for a task image to be unaccessible then is not to be on the booted (system) device, which is the only accessible device at that time.

    To make it easy to avoid this situation, a convention has been established. If a common or task is installed from any LBn:, it is assumed that its common or task image file will be accessible when the system is booted. Any LBn: was chosen as the criteria, because LB0: is the default device for VMR INS and because LB0: tends to always point at the booted (system library) device. Because LBn: will pass this test, users can run multidisk systems if they are willing to assume the responsibility that all such disks will be accessible when the system is booted.

    The MCR command CBD lists checkpointable commons.

12. If the processor on which SAV is executing is not the only active processor, a PROCESSOR x IS NOT STOPPED error message is output for the first such processor, and SAV exits. Note that the CPU identifier x in the error message is a letter (A, B, C, or D).

    CON DISPLAY FULL FOR CP shows the status of all CPUs. CON OFFLINE CPx or CON OFFLINE ALL can be used to take CPUs offline.

13. If any mountable devices in addition to the system device are mounted, a VOLUME STILL MOUNTED ON ddn: error message is output for the first such device, and SAV exits.

    The MCR DEV command can be used to determine which devices have mounted volumes.

14. If any files are still open on the system device, an OPEN FILES ON ddn: error message is output, and SAV exits.

    This is most likely caused by having a print or batch queue still active. Stop the various despoolers and the queue manager.

15. If the system device is mounted, SAV attempts to automatically dismount it. If the dismount succeeds, a couple of different messages can be output depending on whether or not TKTN is installed. If the dismount fails, an ERROR ATTEMPTING TO DISMOUNT ddn: error message is output, and SAV exits.

16. If any task has I/O active, a TASK name HAS OUTSTANDING I/O error message is output for the first such task, and SAV exits.

    The MCR ATL command can be used to identify such tasks.

    The test is made, because such tasks would hang when the system is later booted and, needless to say, the device is no longer active to cause an interrupt.

17. If any task is active and checkpointed, a TASK name IS ACTIVE AND CHECKPOINTED error message is output for the first such task, and SAV exits.

    The MCR ATL command can be used to identify such tasks.

    The test is made because tasks can be simultaneously installed in more than one system image file. Therefore when the other system image runs, it could checkpoint the task into its task image file. When this system image file is then booted, the task from the other system would be simply loaded into main memory and used (or executed) by the system. This would probably cause a very mysterious crash.

18. If any task is found to be connected to an interrupt vector (the $CINT directive), a TASK name IS CONNECTED TO AN INTERRUPT VECTOR error message is output for the first such task, and SAV exits.

    Abort the task or use some other method to cause the task to relinquish the interrupt vector.

An RSX-11M system cannot tolerate tasks being connected to interrupt vectors because the TCB word T.CPCD is used by both SAV and connect-to-interrupt. This is not a problem in M-PLUS, but the problems of tracking down the interrupt vectors (they are not in the system I/O data structures) and bus affinity are severe enough to postpone a general treatment in SAV until a future release (if ever).

19. If any tasks are installed from some device other than an LB:, a TASK name IS NOT INSTALLED FROM AN LB: error message is output for the first such task, and SAV exits.

    The MCR TAS command can be used to determine from which device a task is installed.

    The rationale behind this check is the same as for having checkpointable commons installed from an LB: and suffers from the same problems.

    Note that VMR INS will install tasks only if their task image files are on LB0:.

20. SAV then disables checkpointing in the system.

21. A test is made to insure that accounting is turned off. If it is active, the message ACCOUNTING IS ACTIVE is sent, and SAV terminates.

22. If secondary pool is not completely within the range of memory to be saved in the system image file, SAV will output the message SECONDARY POOL DOES NOT FIT INTO THE SYSTEM IMAGE and terminate. This check is made because many things are now stored in secondary pool (for example UCB extensions.)

23. If any controllers except those for the system device, TI:, MK:, and II: are online, a PROPER CONTROLLERS AND/OR UNITS ARE NOT OFFLINE error message is output, and SAV exits.

    If any units except those for the system device, TI:, MK:, pseudo devices, and RD: are online, the error message is output, and SAV exits.

    In either case, CON DISPLAY FULL will show which controllers and/or units are online.

    CON OFFLINE ALL should leave only the permissible controllers and/or units online. However, because CON OFFLINE ALL suppresses error messages, the system may be having problems during the offline process and you may be unaware of them. In this case, CON should be used in an attempt to put individual controllers and/or units offline. The problems pointed out by the resulting error messages should be corrected.

TI: and the load device are exceptions, because SAV wants to send QIOs to them. Pseudo devices are exceptions, because they are always redirected to real devices, which can be handled normally. MK: and II: are exceptions, because they are integral parts of a system configuration. RD: is an exception, because it must be online to bring anything else online.

24. At this point all normal operations of the system should be quiesced. However, each main partition's wait queue is checked to insure that no task is still checkpointed into its task image. Note that the checkpoint files have already been checked to make sure that they are empty. SAV also looks at the loader task's receive queue and will output a message if anything is queued to the loader.

    All of the necessary checks on the system have now been made. The checks that follow assume that the system is in a quiescent state.

25. If SAV cannot find the name of the required special driver in the table in PSECT DRVTAB, a NOT VALID SAVE DEVICE error message is output, and SAV exits.

    Except for internal errors (the size of one entry in DRVTAB is wrong or a module is missing) and a corrupted $SYSIZ+6, this check should always succeed.

26. If the desired special driver is found, the UMR usage flag is set and the BAE offset is calculated (for RH and DL devices) and the ring and packet pointers are initialized in case the device is connected to a UDA50 controller. The special driver is then initialized to read in the system image file and is then copied into the system image bootstrap if necessary.

27. The structure level of the volume on which the system is being saved is checked. This prevents "old" SAVs from being used in situations that they might not be able to handle. For example, when a disk supports 65K files per volume, it will have a multiheader index file. SAVs that have not been modified to handle this circumstance should not be used on such disks.

28. The label block of what SAV believes to be the system image file (LBN in $SYSIZ+3 and $SYSIZ+4) undergoes several tests. If it cannot be read in, a LABEL BLOCK I/O ERROR message is output, and SAV exits. If it does not pass a few cursory checks on its integrity as a label block (for example, the file has no header), a $SYSIZ DOES NOT POINT AT SYSTEM IMAGE FILE error message is output, and SAV exits. If all the checks succeed, the number of blocks composing the system image file bootstrap minus one is written into the label block as the "transfer address", and the label block is written back to the file.

Altering the transfer address of the file gives VMR and BOO a simple test for determining if a file is a saved or virgin system image as well as for telling them how long the system image bootstrap is.

29. Context switching is disabled so that tasks the Executive starts before SAV restores the T.LBN's won't get into the active task list and then block all tasks except for SAV. The low order bit of T.STAT is set in every TCB except SAV's. Because this is one of the blocking mask bits, all tasks except SAV are prevented from executing.

30. All copies of a multiuser task have a pointer to the read-only part of the "parent" task image. In a running system, P.LBN is the high byte of the LBN of the first byte of the read-only code. P.LBN+2 is the low order word of P.LBN. P.LBN+4 is zero. In a saved system image file, P.LBN is the high byte of a number which when added to the LBN equivalent of T.LBN will yield the LBN of the read-only segment of the "parent" task image file. P.LBN+2 is the low order word of P.LBN. P.LBN+4 is nonzero to indicate that P.LBN contains a relative number.

31. For every installed task, T.LBN, T.LBN+1, and T.ASTL (if a task is not fixed and inactive) or P.SIZE+2 (if the task is fixed or active) is overwritten with the task's File ID. INS copied the File ID from the task image file label block into the task's header.

    If SAV cannot read a task header during this process, a TASK HEADER READ ERROR message is output, and SAV halts the processor. At this point, the system has been corrupted. Not only have all tasks except SAV been stopped, but some of the P.LBNs and T.LBNs have been converted. Only the part of SAV that brings a system up can reconvert those pointers to the form used in a running system. You must delete the system image file and use VMR to apply the appropriate SYSVMR.CMD file to a copy of the corresponding virgin system image file.

32. If SAV was invoked with the /WB switch (write hardware bootstrap), the special driver is initialized to read the system image bootstrap into low main memory and is copied into the hardware bootstrap block (see $BTBLK in the module SAVBOT). The LBN of the system image bootstrap is written into locations 4 and 6 of the hardware bootstrap block and the hardware bootstrap block is written into LBN 0 of the disk.

33. The special driver in SAV is then set up to write all of main memory into the system image file. This includes setting the ring and packet pointers in SAVCM in case the SAV device is the UDA50.

34. SAV calls the Executive (and consequently the appropriate drivers) to put the controllers and units of the system device, TI:, MK:, and II: offline. For several reasons, it is assumed that there will be no delay in this process and that it will succeed (see WIPVC in the module SAVST for the details). The controllers and units for programmable memory boxes are simply marked offline.

35. CO: is redirected to TI: (see below for the special action that is taken if this is not still true when the system comes up) and all terminals are set nonprivileged and logged out.

36. The software volume valid bit is cleared for the system device. After this, SAV cannot issue any additional QIOs because they will be rejected by the driver.

37. If the clock has a CSR address, it is stopped. The processor's mapping registers are saved on SAV's stack.

38. Bytes 0 through LOWSIZ*2 are copied into a buffer in SAV. The system image bootstrap (see $VBN1 in module SAVVB1), which includes the special driver primed to read in the system image file, is copied into low physical memory.

39. The LBN of the system image file (the system image bootstrap) is copied into physical locations 4 and 6 and into physical locations BTADD+4 and BTADD+6.

40. The LBN and offset within the block of SAV's buffer for low main memory are written into physical locations 10 and 12. This allows VMR to find low main memory in the system image file.

41. SAV calls its copy of the special driver to copy all of main memory into the system image file. The LBN of the system image file is found in locations 4 and 6.

42. When the driver is finished copying main memory, it transfers control to location 0, where the unused system image bootstrap resides. Note that at this point the CPU is left in 22-bit mode with D-space, and the unibus map turned off.

IX. What SAV does in Detail (Restoring the System)

It is at this point that control passes to the saved system from BOO, SAV, or from a hardware bootstrap. Note that this point can be entered with the CPU in either 16- or 22-bit mode. If the system has just been booted by the hardware bootstrap, the CPU will be in 16-bit mode. If control got here via the SAV or BOO commands, the CPU will be in 22-bit mode.

1. If the system has been booted by the hardware bootstrap, the boot block (LBN 0) of the disk has just been read into main memory locations 0 through 777. The purpose of this boot block is to read in the system image bootstrap. The boot block was written to the disk by the /WB command.

   The boot block relocates itself to a place in memory that is well above the size of the system image bootstrap. This is currently calculated by adding 1000 to the size of the system image bootstrap. Once the boot block has been relocated, a copy of the special driver is used to read in the system image bootstrap. For MASSBUS, RK611/711, and UDA50 devices, a simplified special driver is used because of size constraints in the boot block (for example, the boot block can only be 256 words long.) These simplified drivers take advantage of the state of the disk hardware left setup by the hardware bootstrap.

2. The system image bootstrap picks up the LBN of the file to read into main memory from locations BTADD+4 and BTADD+6 and deposits them into locations 4 and 6 for use by its special driver in reading in the rest of the system image file.

   If the system image bootstrap had been read into main memory by BOO, the contents of BTADD+4 and BTADD+6 would have been supplied by the module BOTPH2.

   If the system image bootstrap had been read into main memory by the disk's boot block (see $BTBLK in the module SAVBOT), the contents of BTADD+4 and BTADD+6 would have been supplied by the disk's boot block. The LBN of the hardware bootable system is stored in relative locations 4 and 6 of the disk's boot block. When control is transferred to the hardware boot block from the hardware bootstrap, the hardware boot block relocates itself to BTADD and then reads the system image bootstrap into locations 0 to BTADD.

3. The routine $VBN1 is then entered and a number of things happen to get the system back on the air. Figure 3 gives a graphic description of the steps executed in $VBN1. The $VBN1 process involves the following steps:

1. The processor's mapping registers are set up. Kernal APR6 is specifically set up for use by certain special drivers and User APR5 and APR6 are set up to map to SAV itself.

2. Register SR3 is set for 22-bit and UMRs. If a nonexistent memory trap occurs, the processor does not support 22-bit addressing. For M-PLUS V2.0, this is an invalid condition.

3. A special test is made for a DL (RL01/02) device connected to an RLV12 controller. If this test is successful, SAV is running on a PDP-11/23-Plus processor which has 22-bit addressing and a BAE register.

4. $VBN1 now looks at the name of the special driver contained in the driver block further up in the $VBN1 code. If the driver is the common driver (SAVCM) and the boot device controller is an RK611 or a UDA50, then SAV assumes that UMRs are required to read in the image. If the special driver is not SAVCM, then UMRs are used by default.

5. SAV now knows that the boot device controller is either an RH11 (no RHBAE) or an RH70. If the SYSID register does not exist, SAV must be executing on a PDP-11/24 or PDP-11/44 processor, which means that the controller is an RH11 and UMRs are required. If the SYSID register exists, SAV is executing on a PDP-11/70. However, the controller may not be an RH70. SAV then determines which controller is used and calculates the RHBAE offset by testing for the last controller register that responds and subtracting the CSR address from it if the controller is an RH70.

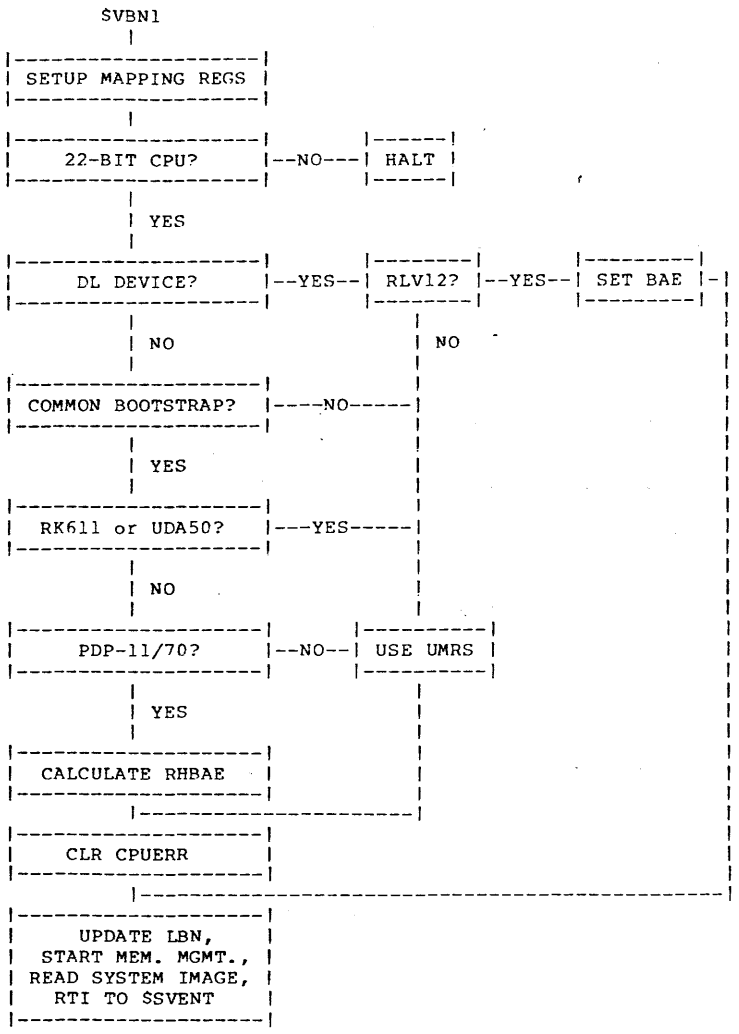
6. If this is a 22-bit CPU and the boot device controller is not an RH70, set register R2 nonzero to indicate that UMRs are required for the special drivers.

7. The CPUERR register is cleared to remove any residual error indicators. Next the LBN of the system image is updated to point to the next area of the image to be read in (currently VBN3).

8. The memory management unit is then enabled (SR0).

9. Control then falls into the system image bootstrap's special driver, which reads into main memory (starting just above the system image bootstrap) the remainder of the system image file.

FIGURE 3 -- FLOW DIAGRAM FOR $VBN1



4. The system image bootstrap then passes control to SAV at location $SVENT in the module SAVE. There the LBN of the system (found in locations 4 and 6), the physical unit number, the physical device name, and CSR address of the load device (passed to SAV in R4, R1 and R5) are saved. Bytes 0 up to LOWSIZ*2 are restored from SAV's buffer, and the mapping registers are restored.

   SAV currently initializes user data space by restoring it from its own user I-space. This implies that SAV cannot be built with user I/D space support unless the code is changed to save and restore both sets of registers.

   When the UMRs are restored, three distinct algorithms may be used. First, if this CPU has UMRs and the saved system had UMRs, then load the UMRs with the values saved and set HF.UBM in the hardware feature mask word ($HFMSK). Second, if this CPU has UMRs and the saved system didn't, then load only the first five UMRs and set HF.UBM. Third, if this CPU has no UMRs (PDP-11/23-Plus), then simply clear HF.UBM.

5. To increase the system's transportability, test the FPP hardware and see if it is present. If it is, clear HF.FPP, otherwise, set the bit. Then test to see if the booted processor has the CIS hardware. Note that this check uses a special stack because the CIS hardware requires a minimum of 64 words of stack space. Set HF.CIS if the CIS hardware is present.

6. If the system is a multiprocessor system, the Interprocessor Interrupt and Sanity Timer (IIST) is interrogated to determine the number of the booted processor. If the IIST exists (if it does not, the booted processor is considered to be CPU A), the processor is mapped to the appropriate processor-dependent, low main memory context. The IIST is marked online, and its interrupt vector is established.

   Note that a dual processor system consisting of CPUs A and B cannot be booted and then run as a dual on CPUs C and D. Each processor requires its own processor-dependent context. The processor finds its context by determining its absolute processor number by interrogating the IIST and then looking in the appropriate part of the CPU partition. A dual consisting of CPUs A and B does not have processor-dependent context areas for CPUs C and D. Therefore the only truly transportable multiprocessor system is either a single- or a quad-processor system.

7. If the booted processor is not the processor on which

the system was saved, SAV's context is logically transferred to the booted processor. This involves actions such as logically making SAV the current task on the booted processor, marking the other processor as stopped, exchanging trap vectors, and the like.

8. If the processor on which the system was saved had a free running clock (a line frequency clock with no CSR), SAV checks for a programmable clock or a line frequency clock with a CSR. If one is found, it is used as the system clock. Unless this is done before a system was saved on a processor with a free running clock, it could not be run on a system without a free running clock, because no attempt would be made to find (enable the interrupts for) another type of clock. If no clock with a CSR is found, it is assumed that the booted system also has a free running clock.

   Note that SAV will hang when it waits for the first redirect of the system device to occur unless the system has a ticking clock.

   If the booted processor has the same type of clock as the processor on which the system was saved, that clock becomes the system clock. If that is not so, it must be one of two cases:

   - The processor on which the system was saved had a line frequency clock, and the booted processor has only a programmable clock. SAV runs the programmable clock at line frequency. Note that if the programmable clock has not been properly installed (had a line frequency signal run to its Small Peripheral Controller slot), it will not tick if run at line frequency.

   - Or, the processor on which the system was saved had a programmable clock, and the booted processor has only a line frequency clock. SAV treats line frequency as 50 or 60 hertz, depending upon the SYSGEN response to the system hertz question.

   Note that all items in the clock queue are scheduled by ticks. Not finding the type of system clock with which the system was saved may cause SAV to change the number of ticks per second for the booted system from that of the saved system image file. SAV makes no attempt to alter the clock queue to reflect any change in the number of ticks per second. All the events already in the clock queue will occur in the correct relative relationship, but the absolute times when they occur will be off by an undetermined factor. However, entries added to the clock queue after the system is booted will occur at the correct absolute times.

9. By examining the contents of the Floating Point Processor (FPP) trap vector and testing for the existence of the PIRQ register, the FPP trap vector is altered to correctly handle the processor's FPP.

10. To increase the system's transportability, the Executive contains a table of CSR addresses. These addresses are typically those of parity memory modules.

    SAV checks to see if those addresses exist in the I/O page of the booted processor. If not, the table is altered to point to a main memory location. This allows the Executive to contain unconditional code that manipulates those CSRs.

11. A series of tests is then made to determine the type of CPU that SAV is executing on. At the present only the PDP-11/74, PDP-11/70, PDP-11/44, PDP-11/24, and PDP-11/23-Plus are fully supported. The CPU model number is stored in the Executive as a decimal value.

12. Main memory is then sized. This consists of writing a zero into every memory location above the area written into by the boot process (so the parity memory bits are set to a known state) and noting where memory first does not exist.

    SAV always sizes memory because after the CPU mapping has been set up, SAV might be able to find memory that the special driver could not when it was DMA'ing in the system image. For example, we might find UNIBUS memory on the PDP-11/44.

    The last (highest) main partition in the system that is not a device common is automatically expanded to cover all memory that is found, and $SYSIZ is updated. The partition should not be expanded if it is a CPU or secondary pool partition.

    If the booted processor has MKA11 main memory, SAV will find only the memory boxes that have their "force panel" switches set to force the boxes' starting addresses to be determined by the control panel's starting address switches. Either the M9312 bootstrap or BOO will have forced the programmable boxes out of the main memory address space.

13. SAV makes sure secondary pool exists and makes sure that it is within the available memory. Note that this routine must precede the one that brings the console and system disk online, since data structures for those devices are created in secondary pool.

14. The system I/O data structures are then searched for a device with the same name, same CSR, and same physical

unit number as the booted device. If such a data structure is not found (for example, the system was booted on DB2:, but data structures exist only for DB0: and DB1:) or, if the booted device is not on the UNIBUS run of the booted processor, SAV attempts to output (the console may not exist or function) a BOOTED DEVICE NOT IN SYSTEM - dd nnn mmmmmm x error message, where dd, nnn, and mmmmmm are the device name, physical unit number and CSR address for which SAV was looking, and x is the name of the CPU on which SAV thinks it is running. The CPU is then halted with a number greater than 120000 in R0.

15. If the load device exists in the system data structures, SAV, if necessary, alters the data structures to show that the port for the device is switched to the booted CPU and checks for the driver being loaded. If the driver is not loaded, SAV attempts to output a SYSTEM DISK DRIVER NOT LOADED error message and halts the CPU with a number greater than 120000 in R0.

    Boot another system and use VMR to load the driver for the system disk.

16. The Unit Control Block (UCB) for the booted device is redirected to itself. Its software volume valid bit is set, and it is marked as public if the system is a multiuser protection system.

    An attempt is made to bring the booted device online, which, if the operation is successful, accomplishes several things. The interrupt vector(s) will be set to point to the driver interrupt entry point(s), common interrupt routine, or Interrupt Control Block(s) as appropriate. If the device can operate on a MASSBUS, it is determined if an RH11 or RH70 is involved.

    To do this, M-PLUS uses a different technique than does RSX-11M. In M-PLUS, the driver has a special entry point for going online, and the data structures show where the RHBAE register should be if it exists. (If the RHBAE register exists, it is an RH70.) In RSX-11M, SAV contains a table of device names that are associated with MASSBUSes and their standard number of CSR registers.

    If the device cannot be brought online, SAV outputs a BOOTED DEVICE CANNOT BE BROUGHT ONLINE error message, and the system is halted with a number greater than 120000 in R0.

    Short of a bug in the system disk driver or a badly corrupted system, this error message should never be output.

17. If a directive partition exists and is not loaded (the APR mapping for the partition in $DRAPR is zero), SAV attempts to output a DIRECTIVE PARTITION UNFIXED OR NONEXISTENT error message, and the processor is halted with a number greater than 120000 in R0.

    This check is accomplished in two ways. If the contents of $SYSIZ are greater than the contents of $DRAPR, the directive common may have been completely read into main memory. Or, the PCB that is associated with the directive common can be found, and the directive common's upper bound can be checked against the contents of $SYSIZ.

18. The system disk is sized by a module stolen from reconfiguration. Sizing allows the system to adjust to the booted device being slightly different from the saved device. For example, BRU can be used to copy an RK06 disk that contains a hardware-bootable saved system image file to an RK07. The RK07 will be hardware bootable after the copying, because the sizing of the system disk will adjust the UCB for the system device to that of an RK07.

    This implies that SYSGEN questions about what type of disk drives are on what controllers are useless except for virgin systems (because virgin systems do not size disks). Actually, they are even useless for virgin systems except for the RK06/07 case. The RK06/07 driver must know which type of unit is being used, because an RK06/07-type bit in the CSR must be properly asserted or deasserted to read or write the unit.

19. Pool is now sized if the system supports the pool monitoring task (PMT).

20. SAV then determines which unit should be CO:, the console. If CPUn is booted, YLn (the nth DL11 terminal interface) is the default console. In a single processor system, this is TT0:.

    When the system is saved, CO: is redirected to TT0: by SAV. If CO: is not redirected to TT0: when the system is booted (that is, someone uses VMR to alter the redirection), SAV attempts to use the unit to which CO: is redirected as the console. This allows a system to be booted on a configuration where TT0: does not work.

    If the device to which CO: is redirected does not make a good console device (for example, it is not a terminal device, or it does not exist in the booted configuration), SAV attempts to use the default terminal.

If no device makes sense as a console terminal, SAV halts the processor with a number less than 120000 in R0 (currently 1).

21. When the console terminal is determined, SAV alters the system data structures to show the console terminal as logged in, nonslaved, and privileged. CO: and CL: are redirected to it, and it becomes TI: for SAV.

22. Because the module SAVINS can potentially queue commands to MCR, a check is made to make sure that the MCR dispatcher is installed. If it is not, an MCR IS NOT INSTALLED error message is output, and the processor is halted.

23. The system ID message is output.

24. The redirection of the pseudo devices SY:, LB:, and SP: to the booted device is accomplished by direct manipulation of the data structures. This must be done so the MCR dispatching action for the RED and MOU commands that follow will operate from the booted disk.

25. Tasks are then "reinstalled." This basically consists of undoing SAV's alterations to the various P.LBNs and T.LBNs and, if necessary, "rebinding" the tasks' Logical Unit Table (LUT) entries (Logical Unit Numbers, LUNs) to the UCB addresses of this system and the tasks' headers to the PCB addresses of this system. (For information regarding the logical structure of the booted disk and task images, see the appendices of the IAS/RSX-11 I/O Operations Reference Manual and the RSX-11M/M-PLUS Task Builder Manual.)

    SAV first finds and validates the home block of the booted device. After validation, the home block is used to find the index file header, which is also validated.

    For each installed task, the task's File ID is retrieved from the appropriate TCB. Given that and the index file header, the file header of each task can be found. The file header is validated and, if any harm has come to it (for example, the task image file was deleted), an error message is output, and the task is removed.

    Given an intact task image file, the label block is read and the LBN of the task image is determined and is rewritten into T.LBN and T.LBN+1 of the TCB.

    If the task has a read-only segment, the P.LBN that points to the read-only segment is converted from a number relative to the T.LBN of the task image file that contains the read-only segment to the LBN of the

segment. P.LBN+4 is used as a flag to determine if the conversion has already been performed.

If the first word of each of a task's LUT entries is not a valid UCB address of the booted system, Assign LUN Directives are done using the static LUN assignments of the task's label block. The resulting UCB addresses for the booted system are written into the first words of the LUT entries.

Note that while any dynamic LUN assignments made by the task are lost, the UCB addresses that SAV derives are the same UCB addresses that INS would use if the task were to be manually reinstalled using the same global logical assignments as occur in the system image file (that is, if STARTUP.CMD does not establish any global logical assignments). Some sort of reinstallation of the task must be done or the system will crash when a LUT's "UCB address" is used as a UCB address by the system. A task can have the "wrong" UCB addresses in its LUT if the task is simultaneously installed in more than one system image file.

If the label block device assignment for a LUN is not for a device that is in the booted system, a (WARNING) NONEXISTENT LUN ASSIGNMENT FOR TASK name error message is output and no assignment is made for the LUN (that is, a zero is written into the first word of the LUT entry). The task will get an error if it tries to use the LUN without assigning a device to the LUN.

There is a similar problem with the mapping windows in the task header. If the task is simultaneously installed in more than one system image file, the PCB addresses in the windows may not be for the booted system. If SAV discovers such a case, it rewrites the addresses to the appropriate PCB addresses of the booted system. The correct PCB addresses are derived from other system data structures (T.PCB and T.PCBV) so even the effect of installing a task with a /PAR=partition-name is preserved.

There is one case that cannot be covered. If the task is active and has dynamically mapped regions, a TASK ACTIVE IN ANOTHER SYSTEM, TASK REMOVED - name error message is output, and the task is removed.

If the partition in which the task should execute has shrunk so much (because of the small amount of main memory on this processor) that the task can no longer fit into the partition even if nothing else is in the partition, a TASK TOO BIG FOR PARTITION, TASK REMOVED - name error message is output, and the task is removed.

If the task image for SAV has been damaged (deleted,

for example), the error message SYSTEM MAY NOT BOOT CORRECTLY is output in addition to the message about removing SAV. If the system does not have enough main memory to allow both SAV and MOU to reside simultaneously in main memory, the system will now hang. This is because the system needs to checkpoint SAV to get MOU into memory to mount the system disk, but SAV has been blocked from checkpointing, so it cannot overwrite the part of the disk where SAV.TSK used to be.

SAV should handle corrupted task images better. There is no need for SAV to worry about deleted task image files if the task is fixed in main memory (for example, the loader).

For each common in the common block directory list that has a task image file, convert its File ID (left in P.FID1, P.FID2, and P.FID3 by INS) into a LBN. This allows BRU and DSC to compress disks containing commons as well as task images.

SAV makes sure that each common task image file is not deleted. If one has been, the system is prevented from trying to checkpoint the common into the task image file and a SYSTEM MAY NOT WORK - CORRUPTED FILE FOR COMMON name error message is output for each affected common.

26. After all the reconverting, the tasks are unblocked, and SAV may no longer be the only running task in the system.

27. The system clock is started after all tasks are reinstalled to prevent the possibility of the Executive finding a clock queue entry for a task to be run on the first clock tick (this could result from a RUN command in VMR). If that occurred, the Executive clears the task's TS.EXE bit, which would make $INSTK look in P.WAIT rather than T.EFLG for the task's File ID. The result would be a bad T.LBN for the task.

    After saving the time the system was booted for RMDEMO, the system clock is started.

28. After reenabling system checkpointing, a check is made to see if the booted processor had enough physical memory to hold all of the system image file's important structures. Important structures are defined to be those that have PCBs and are not device commons. If the check fails, a SYSTEM MAY NOT WORK - LARGER THAN MAIN MEMORY error message is output. Note that secondary pool is checked prior to this.

    The system will work if those structures that were not

read in from the system image file can be logically removed before the system makes an attempt to reference them (for example, unload any loaded drivers that might not have made it into main memory.)

29. Just to give the user a record of what is going on, commands for redirecting the pseudo devices SY:, LB:, and SP: to the system device are queued to MCR. This has the side effect of printing them on the console.

    As an historical note, there is another device name that is treated specially. WK: is meant to be the work file device for such tasks as MAC and TKB. The intent is to identify a LUN that will have heavy I/O usage. Assigning the LUN to a fast device (for example, a fixed head disk) can improve the performance of a system.

    The idea has been partially implemented. VMR automatically creates a global logical assignment for WK: to LB: for all system images. Unfortunately, WK: is not generally referenced in task-build command files. Therefore if WK: is to be used, the task must be installed and the work file LUN must be manually identified and reassigned to WK:. There is little point in doing this, because the work file LUN could just as easily be assigned directly to the real device rather than to WK:.

    There are two reasons for WK: being a logical device name rather than a pseudo device. Pseudo devices must always be redirected to physical devices (with the exception of TI:, which is special cased in the Executive). When the system is booted, there is only one disk device available, the system device. Therefore the pseudo devices are redirected to it. Because it is illegal to redirect devices that are redirected to mounted devices and because the system device is automatically mounted, if WK: were a pseudo device, it could not later be redirected to the real work file device.

    If the tasks that use WK: do runtime LUN assignments, the logical name WK: provides more flexibility than a pseudo device. Each user can independently establish his own work file device by simply making WK: a local logical name. This can prevent running out of space on the one system work file device due to the aggregate demand on that one device of all running tasks that use work files.

30. The system disk is mounted. The defaults that MOU uses can be overridden if the /MOU switch was used when the system was saved. In must cases, overridding the defaults can improve system performance. For instance,

see the discussion of the /LRU and /WIN switches in the discussion of the MOU command in the RSX-11M/M-PLUS MCR Operations Manual.

31. The startup indirect command file is initiated. If the default file specification is used, SAV inserts the hardware name of the boot device to avoid the problem of a global logical assignment for SY: that is not directed to the booted device.

32. Finally, if the system supports the pool monitor task (PMT), PMT is initiated if it is installed.