SHARE

SHARE SESSION REPORT

SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE
	B232	Software Subsystems in MVS	225

PROJECT	SESSION CHAIRMAN	INST. CODE
MVSE MVS Project	Mary Jane Akin	NNL

Northwestern Nat'l Life, Route 3235, 20 Washington Ave. S., Minneapolis, MN 55440, 612-372-5695

SESSION CHAIRMAN'S COMPANY, ADDRESS, AND PHONE NUMBER

James Antognini (IBM) described the elements of subsystems: Identification to MVS, initialization, subsystem interface, subsystem function support, authoirzation, restrictions, command broadcast, notification of other events, and uses and advantages.

A copy of the foils from this presentation and a paper are attached.

3/J/ear/1

SOFTWARE SUBSYSTEMS IN MVS

Session B232, SHARE 61
24 August 1983
New York NY

James Antognini
IBM Santa Teresa Laboratory
555 Bailey Avenue
San Jose CA 95150

17 August 1983, 11:16:01

Abstract

Subsystems in the classical MVS sense are often spoken of but seldom understood even though they are integral to MVS (the master and job entry subsystems). Subsystems are programs started through the master subsystem; they are able to accept broadcast of commands, to be notified of other events and to communicate through the subsystem interface. This paper deals with the elements of subsystems: Identification to MVS, initialization, subsystem interface, subsystem function support, authorization, restrictions, command broadcast, notification of other events, and uses and advantages. Examples are taken from a user-written subsystem and from JES2.

"Subsystem" is a word that has taken on more and more meanings, so that it now may designate merely a complex software program or a sophisticated hardware function. In the classical MVS sense of the word, a subsystem is a program started by means of master subsystem services rather than job entry subsystem services. Also characteristic of a subsystem is the ability to employ the subsystem interface for communication, so that the subsystem can accept its own operator commands and be notified of other events and requests for services. Subsystems have been integral to MVS from the beginning, for the original and still best-known examples are the master subsystem (responsible for such things as starting the master scheduler and the job entry subsystem), JES2 and JES3. Yet the workings of subsystems continue to be an esoteric, even arcane subject.

An MVS subsystem consists of one or more distinct pieces: (1) The subsystem address space (for example, JES2), (2) globally addressable function support routines (as an instance, HASPSSSM) (3) an initialization routine that may be executed shortly after IPL, (4) the subsystem interface (strictly speaking, an operating system component), (5) control blocks created or required by MVS (SSCVT, SSVT, SSOB and SSIB) and control blocks unique to the subsystem and (6) one or more SVCs providing authorized services or carrying out other functions (for example, SVC 100, making job entry subsystem functions available to certain TSO commands). These several constituents will be examined here, but it should be understood from the start that no single one "makes" a subsystem (although the SSCVT control block is the sole indispensable component—an illustrative reflection of MVS's architectural philosophy). A subsystem is better regarded as a collection of capabilities, and a particular subsystem will use various components to realize some of the capabilities.

It is the intent of this paper to put forth the essential features of subsystems thoroughly enough that the seasoned systems programmer will understand their basic workings and could even write his or her own. Two caveats, however: Being so central a part of MVS, subsystems reflect the particulars of the operating system. The details presented here apply to MVS from the base SCP to MVS/System Product (SP) 1.3; later versions of MVS such as MVS/Extended Architecture may well introduce changes. Second, the reader of this paper is assumed to be a systems programmer of some experience who is already familiar with the more common MVS conventions, services and manuals.

Identification of Subsystems to MVS

Subsystems may be identified to MVS in several ways. The basic element in identification is the subsystem's name. The name—one to four alphanumeric or national characters, though a numeric cannot be the first

character—may be placed in the SCHEDULR macro and become part of the SYSGEN process; the name may be inserted into the linklist module IEFJSSNT; or, in an SP1.3 system, it may be defined in SYS1.PARMLIB(IEFSSNxx). These respective possibilities are described in System Generation Reference, in Job Management and in Initialization and Tuning Guide.

The linklist module approach also allows designation of a routine to be executed shortly after IPL to effect initialization on behalf of the subsystem. The PARMLIB approach provides the further capability of passing a parameter string to the invoked routine. The initialization routine, invoked during operating system initialization, would probably do things like preparing the subsystem's control blocks, but it might be responsible for carrying out other tasks such as insuring that required volumes are online. It is even possible that only the initialization function would be carried out: A subsystem as such would never be actually started; rather, the initialization routine would be a way of carrying out certain installation-defined actions following IPL. The subsystem intialization routine would serve, in effect, simply as an installation exit during master scheduler initialization.

Subsystem Initialization

An essential component of a subsystem is a pair of control blocks describing it to the operating system: The Subsystem Communication Vector Table (SSCVT) and the Subsystem Vector Table (SSVT).[1] These control blocks give the subsystem its characteristic features: The ability to use the subsystem interface and, in particular, the capability of accepting direct operator commands (such as JES2's "$" commands) in addition to the MODIFY and STOP commands available to any address space.

Soon after IPL, operating system initialization creates an SSCVT for each identified subsystem. The SSCVT contains the subsystem's name in SSCTSNAM (for example, 'MSTR' for the master subsystem, 'JES2' for a job entry subsystem) and a place, SSCTSSVT, for the subsystem to put a pointer to its SSVT; the SSCVT also contains a fullword (SSCTSUSE) reserved for the subsystem's use, and this might be employed, for example, as an anchor for other control blocks defined by the subsystem or by the installation. Following the building of a subsystem's SSCVT, a routine to perform initialization activities is LINKed to if the subsystem's identification in module IEFJSSNT or in PARMLIB member IEFSSNxx named such a routine. This routine can carry out functions like creating the SSVT and establishing any unique control blocks or other resources, with the SSCVT or the SSVT serving to anchor them on pointer chains; a formatted SSVT would be found through an installation-defined chain (eg, SSCTSUSE),

[1] Unless otherwise noted, control blocks are described in the Debugging Handbook and mapped in the usual system macro libraries.

since the SSVT generally would not be connected to the SSCVT via SSCTSSVT at this stage.

After the operating system is initialized, a console operator may issue the command 'START subsystem_name': The master scheduler will start the subsystem's address space, started task control in the new address space will learn via MSTR's subsystem determination function that a subsystem is being started, and the initiator will use MSTR's converter/interpreter interface to read the JCL procedure from SYS1.PROCLIB(subsystem_name). (The *MASTER* address space is a special case: This will have been started by NIP employing the same converter/interpreter to read the master scheduler's JCL in assembled, link-edited format from SYS1.LINKLIB(MSTRJCL) or, in recent SP1.3 systems, from SYS1.LINKLIB(MSTJCLxx)). Any cataloged datasets specified in such JCL must be described in the master catalog; they will not be found if cataloged elsewhere and a JCL error will prevent the subsystem from being started.

It is up to the subsystem to get global storage for the SSVT, to fill in the SSVT and to point to it from SSCTSSVT.[2] The first two requirements could be carried out by the subsystem routine invoked during operating system initialization, or they could be done by the subsystem address space after it has been started. But the third, connecting the SSVT to the SSCVT, must be done only after the subsystem considers itself ready for work, for once the connection is made, the operating system regards the subsystem as active and will examine the SSVT to see whether the subsystem is supporting functions requested via the subsystem interface.

The SSVT contains several fields. The principal is SSVTFCOD, a 256-byte array that holds values corresponding to function codes 1 through 256. The values are indices into SSVTFRTN, a variable-length array of fullword addresses of the routines actually supporting the subsystem functions. A value of 0 has the special role of indicating no support for that function by the subsystem. On the other hand, a value of, say, 1 in the first byte of SSVTFCOD would denote the first fullword of SSVTFRTN as the address of a routine for supporting subsystem function 1. A value of 5 in the second byte would designate the fifth fullword as the address of the routine for function 2; a value of 0 in the third byte would indicate no support for function 3; and so on. A subsystem may stop support for a function simply by zeroing its SSVTFCOD byte and may later reinstate support by setting the byte back to an appropriate index value. The halfword SSVTFNUM gives the largest value in SSVTFCOD; if a function is requested whose value is found to be larger, the subsystem interface treats the request as an error and so indicates to the requester. The addresses in SSVTFRTN are placed

[2] MVS SP1.3.2 contains routines to build an SSVT and to enable or disable functions in an SSVT. These recent enhancements are described in the logic manuals for that level of the operating system.

there by the subsystem after it has determined by LOADing the support routines (or by some other means) what their entry points are.

The greater part of the 256 function codes will have meanings only to a particular subsystem and requesters of its services. Certain codes, though, have significance to the operating system as well. Function code 10, for example, corresponds to command broadcast, whereby the operating system sends notice of operator commands to subsystems that have asked for this information. Command broadcast is an instance of communication via the subsystem interface and gives subsystems the capability of having native commands. If a subsystem has not initiated or has stopped support for command broadcast, it has available only MODIFY and STOP as communication commands.

Termination of a subsystem may be considered part of initialization in that termination consists of undoing effects of initialization. The least a subsystem ought to do when it terminates normally or abnormally is to zero SSCTSSVT, the pointer to the SSVT; a zero pointer is taken by MVS to mean that the subsystem is not active. The SSVT's storage does not have to be freed if it can be used by the subsystem when it comes up again; the SSVT must be locatable in such an event, and SSCTSUSE might profitably serve as the pointer to control blocks unique to the subsystem and including an alternate pointer to the SSVT. The clean-up process might also reset parts of the SSVT and of those other control blocks, especially if a subsequent initialization were to consist only of pointing to the SSVT from the SSCVT again: Any non-zero bytes left in SSVTFCOD would cause those functions to be supported immediately upon the SSVT's reconnection to the SSCVT.

The Subsystem Interface

The subsystem interface is an MVS technique for making requests to subsystems. The particulars of a request are contained in two control blocks, the Subsystem Options Block (SSOB) and the Subsystem Identification Block (SSIB). The SSOB's main components are the function code for the request (SSOBFUNC), a pointer to the SSIB (SSOBSSIB) and a fullword (SSOBINDV) whose meaning depends on the subsystem and the function; this fullword often serves to point to a function-dependent area that is an extension of the SSOB and gives specifics for the function. The SSIB contains the name of the desired subsystem (SSIBSSNM) and, like the SSOB, has a fullword (SSIBSUSE) reserved for subsystem use. When the SSOB and SSIB are prepared and Register 1 points to a one-word parameter list for the SSOB, the macro IEFSSREQ should be employed to branch to IEFJSREQ, the subsystem interface routine. This routine checks the SSOB and SSIB for correct format, locates the SSCVT for the subsystem named in SSIBSSNM and examines the SSVT to see if the requested function is being supported. For supported functions, the address of a supporting routine is found, registers are set up—Register 14 pointing back to the caller of

IEFJSREQ and not to IEFJSREQ itself—and control passes by branch to the routine.

The support routine does what is necessary and branches back to the issuer of the IEFSSREQ macro. A valuable convention for support routines is that they put their return code in SSOBRETN and always set Register 15 to zero; upon return to the issuer of the macro, Register 15 will thereby indicate the degree of success in the interface:

0 (SSRTOK, an EQUate in mapping macro IEFJSSOB): Everything OK; ie, the support routine got the request
4 (SSRTNSUP): Subsystem doesn't support the requested function
8 (SSRTNTUP): Subsystem is not currently up; ie, SSCTSSVT = 0
12 (SSRTNOSS): Subsystem doesn't exist; ie, there is no matching SSCVT
16 (SSRTDIST): Disastrous error, function not complete; eg, selected byte value found in SSVTFCOD is greater than SSVTFNUM
20 (SSRTLERR): Logical error; eg, bad SSOB format, incorrect length

If the support routine was actually entered (R15 = 0), then SSOBRETN gives indication of success or lack of it in the support routine. Under this convention, therefore, the issuer of the macro must examine both R15 and SSOBRETN, since R15 would indicate the success of the subsystem interface in fielding the request and SSOBRETN would tell how the subsystem support routine actually handled it.

Here is an example of making a call to a user subsystem support routine and then interpreting the results of the call:

```
         LA    R2,LOCSSOB          point to SSOB in working storage
         ST    R2,LOCSSOB@         store SSOB's addr
         OI    LOCSSOB@,X'80'      indicate only pointer
         USING SSOB,R2             symbolic addressability
         XC    SSOB(SSOBHSIZ),SSOB  zero SSOB
         MVC   SSOBID,=C'SSOB'     copy EBCDIC identifier
         MVI   SSOBLEN+1,SSOBHSIZ store EQUated length
         MVI   SSOBFUNC+1,GVDSPRT store EQUated function code
         LA    R3,LOCSSIB          get and store addr of
         ST    R3,SSOBSSIB           SSIB in working storage
         USING SSIB,R3
         XC    SSIB(SSIBSIZE),SSIB  zero SSIB
         MVC   SSIBID,=C'SSIB'     copy EBCDIC identifier
         MVI   SSIBLEN+1,SSIBSIZE store SSIB's EQUated length
         MVC   SSIBSSNM,SUBSYSNM   copy subsystem name
         LA    R4,LOCGVREQ         get and store addr of
         ST    R4,SSOBINDV           function-dependent extension
         ...                       prepare function-
         ...                         dependent extension
         LA    R1,LOCSSOB@         point to pointer to SSOB
         IEFSSREQ ,                go to subsystem interface
         LA    R2,LOCSSOB          point to SSOB again
         B     *+4(R15)            branch to wherever
SUCCESS  B     SUCCESS1            R15=0 from IEFJSREQ
NOSUPPT  B     NOSUPPT1            R15=4, function not supported
NOTACT   B     NOTACT1             R15=8, subsystem not up
BADSUBS  B     BADSUBS1            R15=12, subsystem doesn't exist
DISERR   B     DISERR1             R15=16, disastrous error
LOGERR   B     LOGERR1             R15=20, logical error
SUCCESS1 DS    0H                  interpret SSOBRETN
         L     R15,SSOBRETN        load return code from sppt rtn
         B     *+4(R15)            branch to wherever
         B     EXIT                SSOBRETN=0, close dataset
         B     QUIESCG             SSOBRETN=4, subsystem quiescing
         B     NOBUFFRS            SSOBRETN=8, no CSA buffers free
         B     BADESTAE            SSOBRETN=12, sppt rtn ESTAE failed
         B     NOTAUTH             SSOBRETN=16, sppt rtn not authorized
         B     DSNOTOPN            SSOBRETN=20, dataset not open
         ...
         ...
GVDSPRT  EQU   256                 print dataset
         IEFJSSOB ,                map SSOB
         IEFJSSIB ,                map SSIB
         CVT   DSECT=YES           map CVT for IEFSSREQ macro
         IEFJESCT ,                map JESCT for IEFSSREQ macro
```

The non-zero codes in R15 at return mean that the subsystem interface had trouble with the request. The return codes in SSOBRETN have meanings defined solely in terms of the function and have no significance to the subsystem interface.

Subsystem Function Support

Subsystem function support usually has two components: Routines residing in global storage and accessible by branching from any address space, and the address space program which commonly does the substantive work of satisfying out function requests.

Subsystem support routines receive control via the subsystem interface to provide or initiate the function requested. These routines must be in globally addressable storage, for they may be invoked from any address space. The pageable link pack area is most frequently used, but it is possible for subsystem initialization to copy routines into other global storage (CSA or SQA); in these cases, however, the copied routines cannot employ A- or V-constants unless these are updated by the copying program or by the routines themselves.

Upon entry to a support routine, the following information is available:

R0 = address of the subsystem's SSCVT
R1 = address of the SSOB designated by R1's parameter list at entry to the IEFSSREQ macro
R13 = address of a save area provided by the macro issuer
R14 = address for return to the macro issuer
R15 = address of routine's entry point

(In fact, Registers 2-13 are all as they were at entry to the macro.)

What the support routine does depends, of course, on the function requested, but it will often be desirable for the actual work of the request to be carried out by the subsystem address space rather than by the support routine. Handling native commands, described below, is such an instance, for actually doing what a command asks could be time-consuming or involve a long wait. All a support routine might do is to verify the correctness and legitimacy of a request and then to pass it on to the subsystem address space by a cross-memory post, by manipulation of the subsystem's unique global control blocks, by the PC instruction or by some other mechanism. The subsystem address space would receive notification, find the work and do it. Generally the caller would be notified when the work was complete. (If the subsystem address space does the work, the user would not, of course, be accountable or billable for it unless the request were somehow recorded by the subsystem's support routine or address space. JES2, for example, writes an SMF record for the amount of printing/punching actually done on behalf of a user.)

Certain functions may be requested by one or more tasks or address spaces at the same time; the broadcast of operator commands and notification of other events are cases where requests for the same function may occur more or less simultaneously. It is the responsibility of the subsystem's support routines and address space to maintain the integrity of resources such as control blocks and buffers. If, for example, data such as command strings are to be copied from local areas to global buffers, the use of buffers by subsystem components must be serialized. Furthermore, if sensitive information such as passwords might be in such areas, it would be advisable to get such storage from fetch-protected, non-key-8 storage so as to prevent snooping by programs that are not part of the subsystem.

Subsystem Authorization

A subsystem support routine has only the authorization of its caller, because it is entered by branch; the subsystem interface does not involve an SVC call, so there is no inherent authorization. The support routine will, however, usually require some authorization, since it is typical for the actual work of function support to be effected in the subsystem address space. Notification of that address space may be done by cross-memory post or, alternatively, by manipulation of the subsystem's

own control blocks, and these would presumably be in some system key (frequently, key 0) or at least in a key other than key 8, the ordinary user key (otherwise, ordinary programs could tamper with the subsystem's resources). But since the support routines are generally called by branches from such user programs, there is a need for an authorization mechanism to permit the support routines to do what the user programs cannot achieve directly.

The simplest way to authorization is an SVC issued by the support routines: The SVC might carry out the manipulation of control blocks or issue the cross-memory post. A variant of this approach is an SVC ascertaining that its caller is indeed a legitimate subsystem support routine and then instating job-step authorization, so that the support routine may employ MODESET or other services needing authorization; when the routine has done its work, job-step authorization would be relinquished by reinvoking the SVC to turn off authorization or by the routine doing so directly whilst in key 0. A third way to design an SVC is for it to make requests via the subsystem interface; the support routines thereby inherit the SVC's authorization. SVC 100, employed by the TSO commands SUBMIT, STATUS/CANCEL and OUTPUT, is an example of this last approach; SVC 34 (for issuing operator commands) and SVC 35 (for issuing WTOs) are other instances.

A different technique, available in SP1.3 systems, is to employ the PC instruction to transfer control to routines in a different address space that enjoys greater authorization than the calling address space. The called address space might well be the subsystem address space itself.

Whatever procedure is chosen as a means of authorization must possess carefully devised controls to guarantee legitimate use. The users of the technique must be true subsystem support routines and not bogus routines devised by a user to cause control to return to him in an authorized state or to bring about changes in subsystem or operating system resources that the user could not achieve directly and which the subsystem's designers did not intend. Insuring legitimate use of routines that are to run authorized is a considerable topic, one on which a great deal of effort has been spent both by systems designers and by people attempting to thwart controls. It will not be explored here. It must suffice that the reader is warned that this matter is important and difficult and may constitute one of the most formidable tasks in building a subsystem.

Quite often the substantive work of function support is carried out in the subsystem address space. Performing such work will generally require that the subsystem address space be authorized itself. It will, for example, need to run in a special key to change its global control blocks. It may have to do a cross-memory post of its caller if the caller is waiting for completion of a piece of work, and the subsystem address space may need all manner of other system services restricted to authorized routines. The subsystem address space enjoys no inherent authorization; even though it has been started via the master subsystem rather than a job entry

subsystem, the address space is subject to security mechanisms (eg, expiration dates on datasets, password, Resource Access Control Facility) and integrity mechanisms (enqueues) and has been initiated in user key and problem state and is subject to CPU consumption restrictions (job-step CPU time limits, SRM controls). In sum, a subsystem address space has no privileges merely because it is a subsystem.

Other mechanisms are necessary to confer such privileges. The subsystem's job-step program (that is, the one designated by 'PGM=' in its JCL) might be an authorized program from an authorized library, so that it can use MODESET and other restricted services; or the program might receive control in a system key because it was identified in the Program Properties Table (PPT; consult the Job Management manual for details); or the program might have a unique authorizing mechanism such as a special SVC.

In short, a subsystem requires two things to be effective: Identification as a subsystem and a way of getting control in an authorized state.

Restrictions on Subsystems

Some restrictions apply to subsystems. A couple have been mentioned. One is that a subsystem's JCL procedure must be in SYS1.PROCLIB so that MSTR's converter/interpreter interface can find it. A second is the requirement that datasets in the JCL be identified by UNIT and VOL=SER parameters or be cataloged in the master catalog; datasets dynamically allocated by the subsystem after being started may, however, be identified in other catalogs. Another restriction on allocations is that subsystems, since they are not started by a job entry subsystem, cannot use SYSOUT datasets; if dump information is desired (for example, through the SYSABEND ddname), a dataset must be allocated instead of SYSOUT.

Several limitations apply to subsystem execution. If the job-step program is identified in the PPT, it has to be from an authorized library and it must be invoked in a single-step procedure (otherwise, the program receives control with problem program attributes such as key 8). Subsystem support routines should ordinarily be in SYS1.LPALIB, although they may be copied into any globally addressable storage so long as A- and V-constants contain correct addresses. A number of function codes, listed below under "Events Broadcast to Subsystems," have meanings predefined to operating system components for certain functions and should be employed by subsystems only to support those functions. Furthermore, quite a few function codes—1 through 63 at the time of writing—have specific meanings for MSTR, JES2 or JES3 (consult System Logic Library for details); although other subsystems could probably use the non-broadcast codes without problems, it is more prudent to employ function codes from 256 on down, just as has been the recommended practice for user SVCs.

Support for Command Broadcast

Certain subsystem function codes have predefined meanings to MVS. Probably the most interesting is code 10, command broadcast. The operating system uses the subsystem interface to give control to the support routine of each active subsystem wanting command broadcast (as indicated in the tenth byte of the subsystem's SSVTFCOD). Such a support routine is an exit from SVC 34, so that the routine receives control in supervisor state, key 0, enabled, in the issuing address space (eg, *MASTER*, a TSO user) for all operator commands.

At entry, R1 points to an SSOB, SSOBINDV points to a function extension, and in the extension is a pointer to the command buffer. This buffer can be examined to see if the command belongs to the subsystem, where "belonging" means whatever the subsystem defines that to be: Perhaps commands that have no equivalent in MVS, JES2 or JES3, perhaps commands that begin with a special character like JES2's "$". The following illustration shows how to recognize a subsystem's commands (begun by ".") and how to handle commands that do not belong to the subsystem or that belong but cannot be processed (perhaps because of a shortage of subsystem resources like its own buffers in CSA):

```
         USING SSOB,R1
         L     R7,SSOBINDV              point to function area
         L     R4,SSCMBUFF-SSCMBGN(R7)  point to command buffer
         USING CBF,R4                   map buffer
         LH    R5,CBFCNT                pick up length of buffer
         CH    R5,=Y(CBFL)              standard length?
         BH    NOTOURS                  no, tell MVS it's not ours
         LTR   R5,R5                    zero?
         BZ    NOTOURS                  yes; not ours
         CLI   CBFTEXT,C'.'             command for this subsystem?
         BNE   NOTOURS                  no; not ours
* get length of command string without trailing blanks
         LA    R6,CBFTEXT-1(R5)         point to buffer end
CHKBLANK CLI   0(R6),C' '               current byte a blank?
         BNE   DETORGN                  no; skip ahead
         BCTR  R5,0                     yes; take 1 from length, point
         BCT   R6,CHKBLANK                1 byte back and check more
* R5 = command string length w/o blanks; determine origin of command
DETORGN  CLC   SSCMSCID-SSCMBGN(2,R7),=XL2'0'
         BNE   handle_bad_command       if not 0, not console or TSO
         SLR   R2,R2                    clear R2
         TM    SSCMSCID-SSCSBGN+2(R7),SSCSUSID  TSO user?
         BO    GETASID                  yes, skip ahead
         IC    R2,SSCMSCID+3            no, load UCMID (console id)
         B     GETBUFFR                  skip TSO section
GETASID  ICM   R2,3,SSCMSCID-SSCSBGN+2(R7) load TSO address space id
         N     R2,=X'FFFF7FFF'          turn off SSCSUSID flag (X'80')
         O     R2,=X'80000000'          remember R2 has TSO asid
```

```
GETBUFFR ...                            get CSA buffer for command
         ...
* indicate command is accepted by the subsystem
OURCMD   LA    R15,SSCMSUBC             load EQUated value
         ST    R15,SSOBRETN               and store it
         ...
         ...
* indicate command doesn't belong to the subsystem
NOTOURS  LA    R15,SSCMSCMD             load EQUated value
         ST    R15,SSOBRETN               and store it
         ...
         ...
* indicate command belongs to subsystem but cannot be handled
CANTHNDL LA    R15,SSCMIMSG             load EQUated value
         ST    R15,SSOBRETN               and store it
         ...
         ...
EXIT     LM    R14,R12,12(R13)          restore registers
         LA    R15,SSRTOK               load EQUated value of 0
         BR    R14                      back to MVS
         ...
         ...
CBF      DSECT                          not a standard dsect
CBFCNT   DS    H                        length of entire buffer
         DS    H
CBFTEXT  DS    CL140                    text area
CBFL     EQU   *-CBF
         IEFJSSOB (CM,CS),CONTIG=NO     map SSOB and extensions
```

Whether the command belongs to the subsystem, does not belong or belongs but cannot be processed, R15 is always to be loaded with SSRTOK (EQUated to 0): It is SSOBRETN's value that tells the operating system how the command was handled. When at least one subsystem receiving a command accepts it, MVS does nothing further, and it is up to the subsystem to execute the command and provide information to the issuing console or address space. When no subsystem accepts a command that is not a standard MVS command, the operating system tells the issuer in message IEE305I that the command is invalid. (One can see this happen with JES2 commands before JES2 has been started.) If a support routine accepts a command but cannot actually process it—perhaps because all the subsystem's own command buffers are in use—the operating system sends message IEE707I to the issuer, indicating that the command could not be executed because of a resource shortage in the subsystem.

When a command is to be accepted, the support routine will typically copy the command string and origin to the subsystem's own buffers in global storage so that this information remains available after the return to MVS. Then the subsystem address space is notified in one manner or another so that it can do the actual work of carrying out the command.

Finally SSOBRETN and R15 are set as indicated and the support routine passes control back to the subsystem interface's caller.

This, then, is how a subsystem handles its commands: Its SSVT shows that it is accepting command broadcast and supplies the entry point of the support routine; the support routine is invoked by the subsystem interface with an SSOB pointing to a copy of each command issued in the system; the support routine decides which commands to accept and which to ignore (or to accept with inability to process) and, generally, tells the subsystem address space to carry out the substantive processing of commands that belong to it.

Commands not belonging to a subsystem can be controlled by it, too, because a support routine gets to examine commands before SVC 34 determines their validity as standard MVS commands (eg, START, CANCEL) and executes such commands. By setting SSOBRETN to SSCMIMSG (and R15 to SSRTOK) for MVS commands deemed "undesirable," the support routine would cause them to be not executed; this is, for instance, a way of preventing the subsystem address space from being cancelled. Furthermore, since the routine has access to the command buffer, buffer contents may be changed, as, in fact, the job entry subsystem does to turn short-form replies into standard replies (for instance, '99ANYTHING' becomes 'R 99,ANYTHING'). Obviously, a subsystem support routine should make changes with care, for subsystems receiving the broadcast subsequently and SVC 34 itself will be processing the altered buffer, and the operating system's command capability could be compromised so long as the subsystem is receiving command broadcast.

Events Broadcast to Subsystems

Commands are not the only events that may be broadcast to subsystems. Although details will not be provided here, a subsystem may indicate in SSVTFCOD that it wishes notification of the following events:

End of task (function code 4, SSOBEOT)
End of memory (function code 8, SSOBEOM)
WTO message (function code 9, SSOBWTO)
Operator command (function code 10, SSOBCMND)
Delete operator message (function code 14, SSOBDOM)
Failing START command (function code 32, SSOBCFCD)
Early notification of end of task (function code 50, SSOBFEOT)
Service processor damage (function code 63)

Since the operating system recognizes special meanings for these functions, a subsystem should not employ them for other purposes and, furthermore, should enable them only when it wishes and is ready to support broadcast of information. Thus, the appropriate byte in SSVTFCOD should be set to an indexing value only after all necessary buffers and

other resources are prepared, after the support routine is in common storage (if it has to be copied there) and after the entry point of that routine is placed in the appropriate fullword in SSVTFRTN.

As noted above, it falls to the subsystem to carry out any necessary serialization to protect its resources (buffers, control blocks, etc) that it uses in supporting broadcast functions. The operating system will not do so and it may very well send the subsystem several event notifications in an effectively simultaneous fashion.

Uses and Advantages of Subsystems

Subsystems have uses and advantages that make them attractive from an installation's point of view. One valuable feature is that a subsystem can be started without JES2 or JES3. Consequently, a subsystem can carry out installation-defined functions shortly after IPL and before a job entry subsystem has come up; in fact, a subsystem might carry out those functions and conclude by issuing the operator command to start JES2 or JES3. When the job entry subsystem is down because of problems, an installation-defined subsystem can be a powerful help if that subsystem is capable of things like cataloging and uncataloging datasets, editing JCL, listing VTOCs, locating online datasets and so forth; the list of possibilities is limited only by imagination and programming ingenuity. Anything that might be helpful when jobs cannot be started and TSO users cannot log on is a candidate.

Since subsystems can support their own commands, they can put more operator-friendliness into running MVS. Here, too, the possibilities are limited only by imagination and expertise. One possible application would subsystem commands that cause the subsystem to issue standard but less "friendly" MVS commands; for example, a command like '.ONLINE DASD' could instruct the subsystem to issue the command 'D U,DASD,ONLINE' via SVC 34 after placing proper command and origin information in SVC 34's parameter registers.

Another application of subsystems doesn't require development of a full set of support routines and the job-step program: The SSCVT and the SSVT may serve as anchors for installation-defined resources and control block chains. Such resources might be utilized for billing, system access, job control, security functions, problem reporting and so forth. There is often a requirement for a global anchor for such resources, and whilst there is only one CVTUSER, many subsystems with their associated SSCVTs and SSVTs might be identified to MVS.

Examples of Subsystems

The master subsystem, JES2 and JES3 are the best-known subsystems. There are a few proprietary subsystems such as IBM's Vector Processing Subsystem

(VPSS) and some non-IBM software. IBM's Information Management System/Virtual Storage (IMS) is, however, possibly the commonest product to use subsystem services, though it is not a full-fledged, typical subsystem. Part of IMS installation is definition of the IMS Resource Lock Manager's (IRLM) name as a subsystem. Certain IMS regions will create an SSVT for the IRLM and chain it to its SSCVT, whereupon IMS can function as a subsystem and use the subsystem interface. Specific uses include interception of CANCELs, MODIFYs and STOPs (that is, command broadcast) as well as notification of end of task and end of memory.

In addition to MVS and proprietary subsystems, a few installations have written their own subsystems for internal use. At one site, the subsystem is started just after IPL to carry out chores like issuing operator commands found in a dataset and starting JES2. In another installation, the subsystem is responsible for certain front-end services for terminals. In a third, the subsystem provides VTAM support for certain kinds of printers and also supplies a variety of specially tailored commands to invoke services of help to system operators.

References

Several manuals contain information about subsystems:

OS/VS2 System Logic Library
OS/VS2 MVS System Initialization Logic
MVS Diagnostic Techniques
OS/VS2 System Programming Library: System Generation Reference
OS/VS2 MVS System Programming Library: Job Management
OS/VS2 MVS System Programming Library: Initialization and Tuning Guide
OS/VS2 System Programming Library: Debugging Handbook

Another source of information is the distributed source code for JES2, especially the module HASPSSSM.