27237

DL/CSE/TM30

technical memorandum Daresbury Laboratory

DL/CSE/TM30

SUBSYSTEMS FOR ALL

by

P.E. HAVERCAN, Daresbury Laboratory.

APRIL, 1984

Lending Copy

Science & Engineering Research Council

Daresbury Laboratory

Daresbury, Warrington WA4 4AD

S U B S Y S T E M S   F O R   A L L

P E Havercan,

Science and Engineering Research Council,

Daresbury Laboratory,

Warrington WA4 4AD.

ABSTRACT

This paper describes the Subsystem Interface of IBM's MVS Operating System. It explains what a user should do to define his own subsystem, and how to use Subsystem Control blocks to assist in setting up a Cross-Memory environment. It contains information on how to extract parameters from the SUBSYS keyword of JCL. Some of the applications which use Subsystem Interface at Daresbury Laboratory are described.

CONTENTS — PAGE

1. INTRODUCTION

The Subsystem Interface of IBM's MVS Operating System is one of the most powerful software tools available to Systems Programmers. It was introduced to provide a clean interface between MVS and its spooling task the Job Entry Subsystem. However, the Interface is designed in such a way that customer applications of the system programming type can easily make use of it. Several features new with MVS SP1.3 have increased the usability of the Interface. What has been missing until recently has been adequate documentation on how to write Subsystem applications. The IBM Student Text on Interface Facilities (Ref (1)) discusses Subsystem Interface in a general way, but does not give much practical help. It is hoped that this paper will remedy the deficiency in documentation. It will also describe briefly how Subsystem Interface is to be used at Daresbury Laboratory.

As some misconceptions seem to have been built up about Subsystem Interface, let me first describe what it is <u>not</u>.

- It is not a spooling system, although it is used by JES to provide spooling facilities.

- It is not a method of communicating between jobs or address-spaces, although it provides a structure by which this can be facilitated.

- It does not, itself, use any of the new Cross-Memory Service facilities, although the subsystems which are invoked by it may do so, and are provided with excellent facilities for saving and communicating the required parameters.

So, if it is none of the above, what is Subsystem Interface? It is nothing more (or less) than a set of over forty exit points from strategic parts of the MVS Operating System.

The method of communication with the exits is specified in a formal way so that logical entities called subsystems are invoked by the calls to these exit points. In general, only one subsystem is associated with each call. Nevertheless, some exit points have been deemed so important that a mechanism has been set up for them to "broadcast" the call to all active subsystems. This mechanism is itself a subsystem, called the Master Subsystem.

What facilities does Subsystem Interface give to a System Programmer's application? It gives you, as a System Programmer, the ability to define any or all of the following:

- A global storage area associated with the subsystem, easily accessible by all the routines of the application. This may be regarded as a subsystem-related CVTUSER facility.

- A storage area associated with the subsystem for each task in the system. This may be regarded as a subsystem-related TCBUSER facility.

- A routine entered during initialisation of the Operating System, which may be used for initialising the global storage area.

- An entire Access Method associated with the subsystem.

4) Write suitable Subsystem Function routines.

These Function routines have their entry points specified in the Subsystem Vector Table. The Function routines obey normal OS linkage conventions, viz:

R13 - contains an 18-word savearea into which registers should be saved (using STM 14,12,12(13) instruction)

R14 - contains the return address

R15 - contains the entry point address

R0 - contains the address of the SSCT

R1 - contains the address of the SSOB.

The routine should always return with zero in R15, to indicate that the Subsystem has been invoked. The routine should also set a value in SSOBRETN in the SSOB header. This value (not the one in R15) is used by the caller to determine what action the Function has performed, and is loosely called the return code in some of the documentation (thereby adding to the confusion).

It appears that these routines are always entered in Supervisor state. The protection key in the PSW at entry may not always be the same as the key specified in the TCB. As the latter key is the one used by default by the GETMAIN routine, great care must be taken when acquiring workareas. The protection key of the acquired storage may be specified if a branch entry to GETMAIN is used, but the routine must acquire the LOCAL lock before the branch entry can be invoked.

4. SUBSYSTEM FUNCTIONS

The function which is to be performed by a particular invocation of IEFSSREQ is determined by the function code SSOBFUNC specified in the SSOB. The available functions are listed in Ref (3).

As Subsystem Interface is designed for communication between the Operating System and its Subsystems, only function codes defined by IBM are permitted. There appears to be no range of function codes reserved for customer use, as there is with SVC numbers.

With each function code is associated a particular format of SSOB Extension. This control block passes information related specifically to the function. Each SSOB Extension has a related macro which describes its format. A selection of the function codes and SSOB Extension macros which are likely to be of general interest follow:

Function Code	Related Macro	Description
10	IEFSSCM	Operator Command Processor
05	IEFSSJS	Job Selection Routine
38	IEFSSCI	Converter Exit Routine
39	IEFSSAG	Subsystem Allocation Routine
16	IEFSSDA	Subsystem Open Routine
17	IEFSSDA	Subsystem Close Routine
04	IEFSSET	End-of-task exit
07	IEFSSAL	Subsystem Unallocation Routine
09	IEFSSWT	Write-to-operator exit

DSECTs for the SSOB header and a selection of SSOB Extensions may be generated with the IEFJSSOB macro. The SSOB extensions are selected by specifying a list of the two-character identifiers for the Extensions required. For example, to generate the SSOB Extensions listed above, code:

```
     IEFJSSOB  (CM,JS,CI,AG,DA,ET,AL,WT),CONTIG=NO
```

The majority of Subsystem Requests are sent only to the primary Job Entry Subsystem. Function codes 7, 16, 17, 38, 39 are sent only to the subsystem specified in the SUBSYS DD statement. A few Function codes are "broadcast" to all active subsystems. These include codes 4 and 10 above.

<u>Operator Command Processor</u> (Function Code 10)

This Routine is entered from SVC 34 when a command is entered by the System Console Operator. It enables a subsystem to analyse and modify the text of the command, and to inform SVC 34 whether it is to continue processing the command, or to ignore it.

The SSOB Extension contains a pointer to an area described in some documentation as the Command Input Buffer. This is not to be confused with the CIB which is created later by SVC 34 and chained to a CSCB.

The Command Input Buffer is supposed to be the area containing the command to be issued (i.e. pointed to by Register 1 on entry to SVC 34) but it is in fact a copy of the command in an area which has been padded with at least 16 blanks. The subsystem may therefore extend the command slightly if required. The Command Input Buffer contains the length of the command in the first two bytes, and the command text starts at offset four.

The command in the buffer has been translated to uppercase, and has leading blanks removed (and the leading // if it was entered from JCL).

The Vector Processing Subsystem (VPSS) extends a command which it recognises by adding the text 'F VPSS,' in front of the command, thereby turning it into a MODIFY command. (See Ref (4).) JES2 extends a command beginning with a numeric by adding suitable framing characters to create a REPLY command. (See Ref (5).) These IBM subsystems recognise commands directed to them by the first character in the text of the command, but this is not a requirement of Subsystem Interface.

Unfortunately, JES3 does not process its commands through this interface, as it does not use MCS but does its own console I/O.

This exit could be used to cause certain MVS commands to be ignored, or to redefine the syntax of existing MVS commands, or to create new commands. It can not be used to modify JES commands, because JES will already have seen and analysed its own commands before a user's subsystem receives control.

This routine may set return code 4 in SSOBRETN to indicate that it has handled the input command completely. SVC 34 will then ignore the command. Return code zero indicates that SVC 34 should continue processing the command. SVC 34 will use the (possibly modified) text now in its input buffer.

- A Supervisor Call routine associated with the subsystem, but not requiring any SYSGEN change to the SVC table. It can only be defined when related to the Access Method facility above.

- A mechanism for cleaning up subsystem-acquired resources at both task termination and address-space termination.

- Routines to create new Operator Commands, or to redefine the syntax of IBM-defined commands.

- Limited facilities for defining your own JCL syntax, for use in conjunction with the subsystem-defined Access Method.

Subsystem Interface gives all these facilities, but without the need to modify any IBM code whatsoever!

I will show later that, with a small amount of manipulation, a user's subsystem can also be made to receive the calls normally made to the Job Entry Subsystem, thereby giving the user application access to notification of Job Selection, Jobstep Initiation, SYSIN/SYSOUT Allocation, and Job Termination.

The operation of Subsystem Interface revolves around a control block called the Subsystem Option Block (SSOB) and its Extension. This block is used to pass information to a subsystem exit routine (called a FUNCTION) and to return a completion code, and in some cases a message, to the invoking routine. The SSOB contains a pointer to a Subsystem Identification Block (SSIB).

The SSIB describes the subsystem to which the Subsystem Request is directed. If the SSIB pointer in the SSOB is zero, then a default SSIB pointed to by the JSCB is used instead. The subsystem described by the default SSIB is usually the Job Entry Subsystem. The SSIB contains a four-character identifier by which the subsystem is known. It also contains a fullword SSIBSUSE reserved for the use of the subsystem.

MVS components invoke Subsystem Interface only by branch instructions, not by SVC calls or PC calls. It follows that the Subsystem Function is executed under control of the same TCB, RB, ASID, and protection key as the MVS component invoking the Function. The Subsystem Function routine itself may choose to communicate with another task or address space, but that is entirely at the discretion of the routine, and is not the responsibility of Subsystem Interface.

2. INVOKING A SUBSYSTEM FUNCTION

A Subsystem function is invoked from an MVS Operating System component by means of an IEFSSREQ macro specifying the address of an SSOB. The IEFSSREQ macro invokes the IEFJSREQ routine which does the following:

1) Locates the SSIB from the SSOB or JSCB.
2) Uses the identifier in the SSIB to locate a corresponding Subsystem Communications Vector Table (SSCT).
3) If no SSCT can be found with an identifier matching that in the SSIB, then the subsystem is deemed to be non-existent, and return code 12 is issued.
4) Locates the address of a Subsystem Vector Table (SSVT) from the SSCT.

5) If the SSVT address is zero, then the Subsystem is deemed to be inactive, and return code 8 is issued.

6) If the SSVT address is non-zero, then the function code is extracted from the SSOB and used to index a 256 byte function matrix in the SSVT.

7) If the function index byte in the SSVT is zero, or greater than a maximum specified in the SSVT, then the Subsystem is deemed not to support the function, and return code 4 is issued.

8) If the function index byte is valid, then it is multiplied by four, and the result used to index a vector of routine entry-points in the SSVT.

9) The routine selected by the above process is entered by a branch instruction.

10) The selected routine returns control to the invoker of IEFJSREQ directly by means of a BR 14 instruction.

3. SETTING UP A SUBSYSTEM

In order to set up a Subsystem, it is the responsibility of the author of the Subsystem to perform the following:

1) Create a Subsystem Communications Vector Table.
This is done by specifying the Subsystem name at SYSGEN time (using the PRISUB and SUBSYS keywords of the SCHEDULR macro), or by zapping or recreating the module IEFJSSNT, or by specifying the Subsystem name in member IEFSSNxx of SYS1.PARMLIB. If the last method is chosen, then the PARMLIB member should be selected by means

of the keyword SSN=xx in member IEASYS00, or in response to message 'IEA101A SPECIFY SYSTEM PARAMETERS' at IPL time. The subsystem names generated by SYSGEN are contained in CSECT IEFJESNM in load-module IEEVIPL. The first name in this list is regarded as the primary Job Entry Subsystem.

The SSCT has a fullword SSCTSUSE reserved for the use of the subsystem.

2) Create a valid Subsystem Vector Table.
The format of this control block is described in the IEFJSSVT macro. The SSVT must be addressable from any address-space, so it must be in LPA, CSA or SQA. A 256-byte function matrix determines which functions the subsystem is prepared to support.

3) Chain a valid Subsystem Vector Table to the SSCT.
This can be done in a Subsystem initialisation module which is invoked by Master Scheduler Initialisation at IPL time. The name of the routine to be invoked can be specified in load module IEFJSSNT, or in PARMLIB member IEFSSNxx. (No initialisation routine can be specified for the subsystems specified at SYSGEN time.) If the Subsystem is defined in PARMLIB, then a character string contained in the PARMLIB member can be passed as a parameter to the initialisation routine. (See Ref (2).) The SSVT can also be chained on to the SSCT when the job or system task which is to provide the Subsystem services has been initialised.

An extension to the SSCT can also be created at this time, and its address stored in the SSCTSUSE field of the SSCT. (This area is the global storage area referred to in Section 1.) The SSCT Extension must be in commonly addressable storage.

Job Selection Routine (Function Code 5)

This Routine is invoked by the Initiator to select a job for processing. The subsystem invoked is the one whose name is specified on the START command when the Initiator is invoked. This command is usually issued by JES, and usually specifies the JES subsystem name; therefore the Initiator is usually forced to request jobs from the primary JES.

The Initiator expects a lot of work to be done in this exit: upon return it requires a job to have been selected, a Scheduler Work Area (SWA) to have been built, and all the Job Scheduler control blocks to have been created. JES2 performs most of this work by issuing a LINK to module IEFIB600, the SWA create module. This in turn invokes the Interpreter, which reads previously converted JCL text from the JES Spool dataset, and builds the required Scheduler blocks (such as SIOT, ACT, and JFCBs).

In particular, the JSCB is created at this point, and the address of the SSIB used for this invocation of the Job Select routine is stored in field JSCBSSIB. This SSIB thus becomes the default SSIB for all subsequent Subsystem calls which specify a zero SSIB address in their SSOB. The SSIB is created in the Initiator initialisation routine, and exists for the life of the Initiator.

Converter Exit Routine (Function Code 38)

This Routine is entered from the Converter/Interpreter when it is analysing a user's DD statement containing the SUBSYS keyword. Only the subsystem whose identifier is specified as the first positional subparameter of the SUBSYS keyword receives this function call. The Converter has already transformed the user's JCL

into its own internal text, and the address of the internal text for the whole DD statement is passed in field SSCIINTP of the SSOB Extension. The address of the internal text for the second and subsequent SUBSYS subparameters is passed in SSCISUBS. The routine must return a code in SSOBRETN indicating whether the subparameters coded by the user are valid for the subsystem, and if not whether the job is to be terminated or not. If an error is detected, then an error message can be passed to the user.

Subsystem Allocation Routine (Function Code 39)

This Routine is entered from an Initiator when it is performing allocation for a jobstep containing DD statements with the SUBSYS keyword, or from Dynamic Allocation (SVC 99) for Verb Code 01 and Text Unit Keys X'5F' and X'60'.

When the Initiator is performing SUBSYS allocation, then only one call to each subsystem is performed per jobstep. The allocation requests for the individual SUBSYS DD statements are described in a Subsystem Allocation Request Block (SSARB), and a chain of SSARBs is passed to the Subsystem routine. The address of the first, or only, SSARB is contained in field SSAGARBP of the SSOB Extension.

The SSOB Extension also contains fields in which error codes pertaining to the whole group of allocation requests may be stored, and a pointer to an area which may contain an error message for the whole group. Error codes and message area pointers for the individual DD statements are contained in the SSARB.

The SSARB contains pointer SSAGSSWA to the Subsystem Scheduler Work Area (SSWA) control block for the DD statement. The SSARB is described by macro IEFSSARB, and the

SSWA by macro IEFJSSWA. The SSWA contains the Subsystem name and a number of length/parameter pairs which represent the second and subsequent subparameters of the SUBSYS keyword.

The SSARB also contains pointer SSAGJFCB to the Job File Control Block associated with this DD statement. The JFCB can be modified to provide a DSNAME to be associated with the subsystem dataset, or to pass other information to OPEN.

The SSARB contains a field SSAGSSCM, described as 'Subsystem Information'. This is initially zero, but, if modified, its contents will be preserved and passed to the Subsystem Open Routine (see below). It could be used to pass the address of a control block to be used for communication between Allocation and OPEN.

If this is done, ensure that the control block still exists by creating it in the SWA subpool. This is particularly important when called from TSO Dynamic Allocation. If a task-related subpool is used, then the control block created in the TSO command which performs the allocation may no longer exist in the command which performs the OPEN, but the address will still be passed.

Subsystem Open Routine (Function Code 16)

This Routine is entered from the OPEN SVC when it is opening an ACB for a DDNAME associated with a JCL statement containing a SUBSYS keyword.

If a user program attempts to open a DCB for a SUBSYS dataset, then OPEN detects the condition and creates and recursively OPENs an ACB for the user. The address of a special interface routine is stored in the DCB, which converts the user's QSAM or

BSAM requests into VSAM-like requests which are issued to the ACB created by OPEN. Thus the author of the subsystem needs only to be concerned with ACB processing, and VSAM-like I/O requests.

The purpose of this routine is to supply the address of a routine which will be entered by VSAM macros such as 'GET RPL=xx' and 'PUT RPL=xx'. When the routine receives control SSDADEBP contains a pointer to the DEB created by OPEN to be associated with the ACB being opened. The DEB is not complete - in particular the pointer in it to the current TCB has not yet been set - nor has it been placed on the chain of valid DEBs for the current TCB. It does, however, contain a pointer to the ACB being opened in DEBDCBAD. The ACB should be located via this field, and then the address of the subsystem access method routine should be stored in ACBINRTN. However, if it is desired to store the address of the ACB in any other control block, then the ACB address should be located from DXUDCBAD (in the OPEN/CLOSE/EOV workarea), as the ACB pointed to by the DEB is only a working copy, which will be copied back into the user's storage at the end of OPEN. Addressability to the O/C/EOV workarea may be acquired by noting that the JFCB whose address is passed in SSDAJFCB is actually embedded within the O/C/EOV workarea.

The subsystem access method routine is the interface by which the user will transfer data to and from the subsystem. When it receives control (which will be after OPEN is complete, and when the user issues an I/O request to the ACB or DCB) then Register 1 will contain a pointer to the user's VSAM RPL and Register 0 will contain a function code value. The function code in Register 0 may be a standard VSAM function code, or can be any code which the subsystem is prepared to handle, with any meaning assigned that the author may wish. The access method routine should transfer data as indicated by the RPL and the function code, then return feedback information in RPLFDBK. Register 15 should be set appropriately before returning. See Ref (6) for the settings in RPLFDBK and return codes expected from VSAM macros.

It is possible to specify a Supervisor Routine which the Subsystem Access Method Routine may use to perform functions which require supervisor state or key-zero privileges. (The Access Method Routine may not use MODESET to acquire these privileges, because the invoking program is not usually authorised.)

To provide access to a supervisor routine, save the address of the routine in a fullword. Then store the address of the fullword minus four bytes in DEBAPPAD of the DEB being processed by OPEN. This linkage is defined because JES2 specifies the same address in both ACBINRTN and DEBAPPAD. The first instruction of the Access Method Routine branches over an address-constant containing the address of the JES2 Supervisor Routine. (See Ref (5).) When the Access Method Routine wishes to enter the supervisor routine, it must issue SVC 111 with Register 1 pointing to a valid RPL, or containing the complement of the address of the ACB.

The supervisor routine receives control with registers 0 and 1 preserved. Register 2 contains the address of the DEB, and Register 5 contains the address of the RB of the invoker of SVC 111 (not the current RB). Registers 3 and 4 contain the CVT and TCB addresses, as usual. Register 15 contains the entry point to the supervisor routine, and Register 14 the return address. Registers 6 to 13 are unpredictable.

The Subsystem Open Routine may use field DEBIRBAD to pass information (e.g. the address of a communication area) to the Access Method Routine.

The SSOB Extension for OPEN also contains a pointer to the JFCB and pointer SSDASSCM to 'Subsystem Information'. This is the value stored in field SSAGSSCM by the Subsystem Allocation Routine.

<u>Subsystem Close Routine</u> (Function Code 17)

This Routine is entered from the CLOSE SVC when it is closing an ACB associated with a SUBSYS DD statement. It permits the subsystem to clean up any resources acquired by the Subsystem Open Routine. The same parameters are available as for the Open Routine. However, note that the ACB pointed to by the DEB is now the user's ACB, and not the copy maintained by CLOSE. If modifications are required to the ACB, then use the copied version, pointed to by DXPDCBAD, as this will overlay the user's ACB at the end of CLOSE.

<u>End-of-task exit</u> (Function Code 4)

This is one of the function codes which is "broadcast" to all active subsystems by the Master Subsystem (Module IEFJRASP - See Ref (3)). It permits a subsystem to note that a task which may have used subsystem resources has now terminated, so that those resources may now be released.

Note that because this routine is entered for ALL tasks, an efficient method of deciding whether the subsystem is concerned with the terminating task must be used.

A flag is set in the SSOB Extension to determine whether the task is terminating normally or abnormally.

The routine will probably be required to duplicate some of the functions of the Subsystem Close Routine, as there may be conditions where Close has been unable to clean up due to a corrupted ACB or DCB.

Subsystem Unallocation Routine (Function Code 7)

This Routine is entered from the Initiator when it is deallocating a DD statement containing a SUBSYS keyword. It may be used to release any subsystem resources acquired by the Subsystem Allocation Routine.

This function code is the same as that associated with SYSIN/SYSOUT deallocation, normally handled by JES, so some of the fields in the SSOB Extension seem meaningless for SUBSYS datasets.

Unlike the Subsystem Allocation Routine, this routine is entered once for each SUBSYS DD statement.

Write-to-Operator exit (Function Code 9)

This Routine is entered from SVC 35 before displaying an Operator message on the System Console. It enables a subsystem to analyse and modify the text of the Operator message, and to instruct SVC 35 whether to display the message or not.

The available documentation (Ref (3)) seems to imply that this function code is one of those "broadcast" to all active subsystems by module IEFJRASP. Thus, at first sight, it would seem possible to use this subsystem routine as a substitute for the WTO user exit routine, IEECVCTE. However, this is not possible. SVC 35 uses zero for

the address of the SSIB, therefore only the task's default subsystem (usually JES2) receives the subsystem call. The broadcast routine is entered for some WTOs, but only for those tasks which have the Master Subsystem as their default subsystem. This includes Master Scheduler tasks, but excludes all batch jobs, TSO sessions and started tasks (including Initiators).

The Subsystem Function routine is invoked after the WQE has been constructed. The exit has access to the WQE through field SSWTWQE in the SSOB Extension. The WQE is described by macro IHAWQE. Experience has shown that the text of the message being processed does not begin at WQETXT as the macro suggests, but at WQETXT+1.

5. THE SUBSYSTEM AFFINITY TABLE

Once a Subsystem has been defined, it has associated with it a slot in the Subsystem Affinity Table (SSAT). Conceptually, this is a table of fullwords chained off each TCB. Each fullword in the table is associated with one subsystem, and is reserved for its exclusive use. A subsystem routine may use the SSAFF SET macro to place an entry in the table, and the SSAFF OBTAIN macro to extract data from the entry. For further information see Ref (2). The entry in the SSAT, or a control block chained from it, is the task-related storage area referred to in Section 1.

6. HOW TO RECEIVE THE JES SUBSYSTEM CALLS

It was stated earlier that the majority of Subsystem Interface requests are directed to the Job Entry Subsystem. This is true in practice because of the way that JES starts Initiators. When, for example, JES2 starts an Initiator, it does so by issuing the following command internally:

```
S INIT.INIT,,,JES2
```

The 'JES2' parameter on the START command is used by the Initiator to determine from which subsystem it is to select jobs. JES3 does not specify its name on the command; it leaves the Initiator to assume the default of the Primary JES, which JES3 always is. It issues:

```
S INIT.groupname
```

where 'groupname' identifies the initiator group. If we were to issue the command:

```
S INIT.INIT,,,USER   or   S INIT.groupname,,,USER
```

then the Initiator would attempt to communicate with Subsystem 'USER'. But Subsystem Interface already gives us the facility to modify commands before they are interpreted! We can use a Function Code 10 routine to check for a Start Initiator command and change the JESx parameter to be our own subsystem name, or to add it if it is omitted. Now our subsystem will receive a Job Select Request (Function Code 05) from the Initiator. Unless our subsystem is extremely clever, it is unlikely to be able to supply a job to the Initiator itself. But what it can do is use an IEFSSREQ macro to request a job from the real JES subsystem. Upon return, JES will have created the JSCB for the job, and put in it a pointer to the SSIB used to invoke it. If we wish to continue to receive subsystem calls from the job, then we must reset JSCBSSIB to point to a SSIB identifying our own subsystem as the default one for the job.

A slight problem arises at End of Task. JES expects the default SSIB to contain a pointer in SSIBSUSE to one of its own control blocks. Fortunately there is an Early End of Task Exit (Function Code 50) which our subsystem can use to restore JSCBSSIB before the JES End of Task routine receives control.

Another problem arises because, when allocating SYSIN and SYSOUT datasets, JES inserts its own subsystem identifier into the SIOT for the dataset, and this identifier is used by OPEN and CLOSE. If it is to receive the OPEN and CLOSE calls, then our subsystem must update the SIOT in its Allocation routine. The field in the SIOT to be updated is pointed to by SSALSSNM.

7. USING CROSS-MEMORY SERVICES WITH SUBSYSTEM INTERFACE

Although Subsystem Interface itself does not use Cross-Memory Services, it certainly provides an appropriate vehicle for their use. Assuming that you wish to provide a Cross-Memory Service which is to be available to all tasks (i.e. a service which has acquired a System Linkage Index and has an Authorisation Index (AX) of one) then you are faced with the problem of where to store the Linkage Index, and any Program-Call numbers which may be generated from it.

If your service is a formal subsystem, then you will obviously set up your Cross-Memory environment in the Subsystem Initialisation routine at IPL time. You can then generate an extension to the Subsystem Communications Table, and store in it the Program-Call numbers which identify the routines of your Cross-Memory Service.

It will usually be appropriate to make your service an Access Method, so you will use the Subsystem OPEN routine to provide the address of an Access Method routine which will issue the Program Call instructions. You can pass the address of the SSCT Extension to the Access Method routine by saving it in DEBIRBAD in the user's Data Extent Block.

8. SUBSYSTEMS AT DARESBURY LABORATORY

Although Daresbury Laboratory has not yet migrated to MVS as a production system, an extensive development programme is under way in which Subsystem Interface is playing a large part. The following subsystems are being used, being tested, or are under consideration:

OPER - A subsystem to handle locally-defined operator commands.
JTMP - Job Transfer and Manipulation Protocol.
XSRV - Transport Service.
FTP - File Transfer Protocol.
ARCH - Backup and Archive System.

OPER at present defines two new operator commands: MSG and BRDCST. These commands take the text entered on the command and reformat into a SEND command to send a message either to a single TSO user or to all TSO users.

It also intercepts the DISPLAY command and permits a new operand of SUBSYS to display the characteristics of all subsystems on the SSCT chain. If the operand of DISPLAY is D, then an improved routine to display SYS1.DUMP titles with their times

and dates is entered.

The Subsystem Initialisation routine of OPER is used as a general purpose System Initialisation routine. For example, it is used to define the name of the Primary JES either by reading its name from the IEFSSNxx member which defines OPER, or by requesting it from the operator. This permits multiple test versions of the Primary JES JCL without the need to maintain multiple copies of module IEEVIPL (containing table IEFJESNM), which is IBM's suggested method.

JTMP will be the only method of Remote Job Entry at Daresbury. It is part of the ISO Open Systems Interconnect communications model which is being implemented to provide Networking between the major University and Research Establishment sites in the UK. It is required to keep track of all the jobs which have been entered into the system, therefore it needs all the Job Management subsystem calls which are normally seen only by JES.

XSRV, or Transport Service, is to be another part of the OSI system. It will enable any user program to establish a call to any node on our open-ended X25 network, or to another jobstep task in the same machine. The network access will be through VTAM, but the end user will not need to code any VTAM calls. Transport Service will appear to the user as a Sequential Access Method, and will convert the user's requests to VTAM and X25 requests.

FTP will be a similar service to transfer complete files across the Network, and ARCH is to be a subsystem implementation of Daresbury's existing Backup and Archive system.

9. DEFICIENCIES IN SUBSYSTEM INTERFACE

Subsystem Interface obviously has many benefits to bestow. There are a few very minor defects or desirable enhancements, which I list below. I also suggest circumventions for these deficiencies:

1) The format of the SYS1.PARMLIB(IEFSSNxx) member:

```
subsystem-name,init-routine,parameters
```

requires the 'parameters' field to be enclosed in quotes if it contains special characters, even if the special characters are only commas.

2) The above definition cannot be continued - all the parameters must be contained in one card-image.

A possible circumvention of the above two problems is to regard the 'parameters' field as defining the name of an additional member of SYS1.PARMLIB. The IBM service module IEEMB878 can then be used to read records from the defined member, which may then contain many parameters in any format defined by the subsystem. The IEEBM878 routine can be used from within a Master Scheduler Task (such as a Subsystem Initialisation Routine) to read card images from SYS1.PARMLIB. Before calling the routine, Register 1 must point to a twenty-byte area containing:

+0 - flag byte, set to X'80' to close dataset, X'00' otherwise

+1 - console id, extracted from JSICONID in IEFJSIPL

+4 - address of an 80-byte buffer to contain the PARMLIB record

+8 - eight-byte name of PARMLIB member to be read

+16 - fullword area used by IEEMB878, initially zero.

The routine sets a return code zero if a record is read, and four at end-of-file. Other return codes indicate errors.

3) No SSOB function codes are set aside for customer use. This would be of use when a user exit routine wished to communicate with a user subsystem. Such function codes should ideally be "broadcast" to all active subsystems.

There is no direct circumvention for this problem, except to pick a high value for a function code and hope that IBM will never get around to using it. Note that, although the SSOB defines a two-byte field for the function code, function codes higher than 256 cannot be used without a modification to the Subsystem Vector Table format, as the Function Index Matrix in the SSVT is only 256 bytes long.

4) No subsystem call exists from the DUMP routine. This would be desirable if a subsystem wished to format its control blocks in its user's dump.

A circumvention for this deficiency is to issue a subsystem request from the dump user exit routine IEAVADUS. This has been made to work at Daresbury by selecting function code 255 to mean a dump request. The Abdump Parameter List passed to the user exit is passed to subsystems as the SSOB extension. The dump request is sent to the MSTR subsystem after first modifying the function index for function code 255 in the MSTR Subsystem Vector table to be the same as the function index for End-of-Task. This ensures that the dump request is also broadcast to all active subsystems.

5) The TSO ALLOCATE command does not support SUBSYS allocation.

A new TSO command processor, named SUBSYS, has been written to perform this function.

6) If a task has not issued a SSAFF SET macro, then field TCBSSAT is not zero, but points to a Global SSAT somewhere in the Nucleus. The address of this SSAT is not stored in any conveniently accessible location. Therefore it is impossible in, say, the subsystem end-of-task routine to determine rapidly whether a given task has an associated SSAT, without issuing a SSAFF OBTAIN macro.

A circumvention for this problem is to copy the value from TCBSSAT from the current TCB into the Subsystem CVT Extension during the Subsystem Initialisation Routine. The current TCB during this routine is the Master Scheduler Task, and it may be presumed to point to the Global SSAT as the Master Scheduler is unlikely to have issued a SSAFF macro during its initialisation phase.

7) The SSAFF routine acquires and releases the LOCAL lock. If the caller of the routine owns the lock on entry, he will not own it on exit.

The circumvention is not to issue SSAFF when owning the LOCAL lock. Instead, the Subsystem Affinity Table entry should be made to point to a new control block which can be manipulated under control of the LOCAL lock.

10. REFERENCES

(1) A Guide to Using MVS Interface Facilities (Student Text) (SR20-4675)

(2) OS/VS2 MVS System Programming Library: Job Management (GC28-1303)

(3) OS/VS2 System Logic Library (Vol. 6) (LY28-1079)

(4) Microfiche listings for Vector Processing Subsystem (SJD2-6265)

(5) Microfiche listings for MVS SP JES2 V1R2.0 (LJB2-9520)

(6) OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide (GC26-3838)