NETWORK OPERATING SYSTEM

E. Perotto, CNUCE, Pisa*

*Associate at CERN, Geneva
September 1983 - October 1985

DD-jt
NETWORK OPERATING SYSTEM

Introduction

This operating system was developed in 1981 to support the network of SBCU’s (Sensor Based Control Unit - IBM 5098-N5 RPQ D08116) used at the Stanford Linear Accelerator Center to interconnect the IBM 3081 VM/SP to several DEC VAX 780’s, using Long Line Adapters (LLA) connected to SBCUs through a coaxial cable. The speed of transmission of this device is 2.5 Megabits/sec for distances in the range of a mile, the fastest data communication device IBM has ever offered.

The SBCU’s were driven by the LLA program, a file transfer process written in FORTRAN and Assembler by Keith Rich, which could handle only one file at a time. There was also the need to transfer files in both directions to different locations at the same time, requiring a multitasking environment.

The same requirements had to be satisfied at CERN in 1984 when the CERNET Transport Manager was to be transplanted from MVS to VM. The EXCP I/O interface and the Windows inter-process communication had to be provided to recreate the MVS environment to support the TM. Some enhancements have been added to the VMCF interface to accommodate the Windows. And an additional layer of software and a macro library are now available to emulate MVS Windows.

Definitions:

MODULE: one or more Control Sections (CSECTs) assembled together from the same source file of type ASSEMBLE. A Module starts at a page boundary to take advantage of the coincidence of the last three digits of the LISTING addresses; it is either a TEXT file on disk or a member of a TXTLIB.

TASK: this is a process executing mainly inside one Module, and which may call routines residing in other Modules. Its execution may be suspended by calling the WAIT Function, and can be resumed by the system after some event related to it (POST Function, not accessible by Tasks), or by another Task (WAKEUP Function). Its basic elements are: a 16-Byte Task Descriptor Element (TDE) and its associated Savearea. A Task has a one to four letter Taskname, an Execution Priority (0 highest, 255 lowest), and a Taskid (1-255).

RUNNING Task: the Task which is in execution. It is selected for execution by the Dispatcher according to its priority.

TASK DESCRIPTOR Element (TDE): a 16-byte memory element consisting of a pointer fullword, a Task Name field (4 bytes), a Savearea pointer fullword, a priority code (1 byte), a spare byte, a
TASKSTAT byte, and a TASKID byte. It is used to describe and identify a Task, and is addressed by R10.

EVENT CONTROL BLOCK (ECB): an 8-byte element composed of two fullwords; the first is the ECB itself, the second is the Task Descriptor Element address of the associated Task.
Multitasking Supervisor

Structure

The structure of this Operating System is similar to RSCS, but it runs together with CMS, taking advantage of its File System and of all its resident or transient commands through the SVC interface. CMS actually runs ‘under’ the Network Operating System (N.O.S.), as part of its tasks, used like a library of resident routines. The N.O.S. intercepts I/O and external interrupts before CMS and dispatches the relevant Tasks, which then may call CMS when necessary (e.g. for Disk I/O or to call transient modules). A task retains control of the CPU until it returns to dispatcher. There is no time slicing. The operation of the different tasks is sequential.

The Operating System itself is independent of the particular data communication devices used. Only the I/O Driver Modules need to be designed to handle the specific hardware.

All functions of the Multitasking Supervisor are contained in the module MSUP, loaded at 20000 hex. This module occupies two pages (20-21). The first page is addressed through R11 by all tasks, the reason for sharing this page, is that it contains the common area NETCOM just at the beginning, and the branches to the Supervisor routines are almost direct, without saving or loading any base registers. This saves unnecessary overhead, and for the same reason, supervisor routines do not save registers.

Supervisor routines may be called from two levels:

1) System task level in which registers are not saved, and R11 is assumed to contain the MSUP base address (Hex 20000);
2) User Task level in which all registers are saved in the user savearea (whose address is presumed to be in R13) and R11 is loaded with the MSUP base address. On return all the registers from R2 to R14 are restored, R15 contains the Return Code, and R0 and R1 contain Results.

User Task level calls to MSUP pass through the User Interface, which consists of Macros: FUNCTION, HLINT and RETUSR. These calls are counted and traced. User level code may be written in any high level language which follows FORTRAN linkage conventions.

System Initialization

There are two entry points: INITIAL entered by CMS when the system is linkedited to work alone, and MSINIT called by the User Program when the system is linkedited together with it as a set of subroutines. Both entry points call INITRTN which initializes free memory element queues, the program check, I/O and external interrupt handlers, then generates the first Task (‘INIT’). This in turn calls the initialization entry points on the other modules: CONS, MSG, VMCF, SBCU or CERNET, and RDR in that order, if everything goes well task INIT terminates itself.

The initialization process has to be a task to start other tasks because of the need to wait for response from the new tasks. The initialization routine in each module is executed as part of the INIT task, and starts one or more task per module, the functions of these modules are:

CONS Module - Tasks CONS and VCON:
  to intercept console attention interrupts, issue a Read, direct the operator’s command to the relevant processor depending if NOS, CMS or User command. The VCON Task receives SMSG commands and executes them with the same code as the CONS Task.

CMD Module:
  to process NOS commands. This module was part of CONS module in the SLAC version.

MSG Module - Task MSG:
  to receive message requests from other tasks, decode the message request to a readable message and send it to the destination, which may be the virtual console or a user virtual machine.
VMCF Module - Task VMCF:

to intercept external interrupts with code x'4001' (VMCF), to decode the VMCF header, to
copy part of it to the destination VMCF Link Table and to wake up the associated task, or to
start a new task.

CERNET Module - Tasks MC1R, MC1W, MC2R, MC2W...:
it is executed after the device end interruption from the Modcomp front-end, to start the next
I/O on the receive or transmit side, and to alert the tasks when the I/O operation is finished.

System Termination

This routine resets all the interrupt traps, and returns to CMS. It is used as the final part of the
END Task after a Program Check, an ABEND, or an EXIT command from console.

It contains the CMS Entry Point (now at 2038A), where one can branch by hand to go back to
CMS without reipling it, and without destroying the memory. The CP commands to issue are:

STORE G11 20000
STORE PSW FF000000 2038A
BEGIN

Initialize the base register
Set the branch address
Go

to produce a formatted dump the commands are:

STORE PSW FF000000 1
BEGIN

Set an invalid address
Go

Memory Management

After the two pages occupied by MSUP, three storage pages are organized into three queues of
small memory elements:

256 elements of 16 bytes, used as Task Descriptor Elements
32 elements of 32 bytes, used as Message Elements
106 elements of 96 bytes, used as Link Tables

Their space is reserved by the POOL macros, whose parameters COUNT and LEN may be easily
changed to increase system capacity. These Free Elements must be requested by calling GET16,
GET32, GET96 routines, and released with REL16, REL32 and REL96.

GETxx routines give the address of the memory element in R1 with CC = 2 if a free element was
found, otherwise CC = 0.

There is a couple of routines for variable length memory requests: GETBUF/RELBUF.
GETBUF looks at the FREQUEUE pointer in NETCOM, if it is zero a buffer of requested length + 8
is acquired through SVC 4, the first two words being used for chain pointer and length. If the queue is
not empty, a search is made for the first fit and the buffer is dequeued. RELBUF just requeues the
buffer in front of the free queue. An internal free queue is used so the buffers are not released for CMS
because of the heavy overhead of the SVC 4 to get them again (the GETMAIN/FREEMAIN couple
do calls is around 10 msec CPU on a 3081).

We choose to use the first fit method in order to lower CPU overhead, since the buffer usage
normally lasts only for milliseconds, an inefficient use of memory is not dangerous, especially with
virtual memory. The first fit method, together with the requeuing of last buffer in front of the queue,
gives us a better chance to reuse the same memory page which was locked in memory by the last I/O
operation. Also if the size of the buffer varies: if it decreases first fit gives you the last buffer; if it
increases a new buffer is acquired, and the shorter ones are pushed down the queue. The costly SVC 4
is used only at startup time and at open link time if there are not already buffers of the required size.
TASK creation

There are two entry points for that function: FIRSTASK to create the first Task, and start the dispatcher, in that way really entering the multitasking environment, and CRETASK to create a new Task from another Task, or from the system.

FIRSTASK wants the Task Name in R0, sets the Task execution address to the return address, clears R3-R10 and saves all registers in the Task Savearea acquired by GETMAIN (SVC4).

CRETASK wants Task Name in R1, new Task execution address in R2 and savearea address in R3 if any. If R3 = 0 the savearea is acquired with GETMAIN.

A 16-byte Task Descriptor Element is initialized and queued at end of the Task Queue chain (pointed by TSKQUEUE in NETCOM). The calling Task’s savearea is modified to have the new Task’s TDE address in R1. The new Task’s savearea is initialized with a copy of the calling Task’s R2-R12, with R14 pointing at its own entry point, R0 loaded with new Task Name and R1 pointing at the caller Task’s TDE (may be used for WAKEUP). At this point a branch to the Dispatcher is made, giving the caller Task a chance to be executed before the new task, so that a batch of new tasks may be created at the same time. If the calling Task wants to wait for the successful initialization of the new task, it has to call the WAIT function, and the new Task should Call WAKEUP for that Task when it is ready.

TASK termination

KILLTASK is the entry point to be called to terminate Task execution, free the 16-byte Task Descriptor Element and Task savearea. For now the savearea is not released for debugging purpose, being it very small (x'70' = 112 bytes) it should not do harm to the system. This routine wants R0 pointing to the TDE, if R0 = 0 it means the Running Task is terminating itself, in this case the routine returns to the Dispatcher.

TASK synchronization

DECLCECB and DROPECB are two entry points to add and delete an Event Control Block address to or from a list compatible with the WAIT macro of CMS, although the macro is no longer used.

DCLDEV is a higher level entry point called to declare to the system a new device, its Device table is queued to the Device Table chain, pointed by DEVQUEUE in NETCOM. This chain is scanned at interrupt time to locate the Device Table associated with the interrupting Device and its Event Control Blocks.

DCLEXT is called to declare to the system a new External Interrupt Code to be handled, with the address of the External Interrupt Table in R1. There is one External Interrupt Table for each 16 bit interrupt code which also contains the associated ECB.

QUEUE/DEQUEUE are called to queue a Task to a specified queue associated to a non sharable resource, like a physical device, such queues are not anchored into NETCOM, but rather to Device Tables or Link Tables. Every time an operation is terminated on a device or on a link, the first Task in the associated queue is requeued to the Task Queue (the Dispatcher queue), so that the task is restarted. For instance: the routine DSIO in the SBCU Module checks if the Device is busy, if yes the Task queues itself to an anchor into the Device Table (DEVWAIT); when the current I/O operation is concluded the running Task calls DEQUEUE with the anchor address in R1, thus restarting queued Tasks, if any.
TASK execution

The WAIT function is the entry point used to control the execution of the Running Task. WAIT routine is called without parameters, only the return address must be in R14 as usual, the Task Descriptor Element is requeued at the end of the Task Queue and is updated to reflect the wait status: the WAIT bit is set and the RUNNING bit is reset in TASKSTAT. Any event may wake up the Task, but each task should wait for a single event, so that there is no doubt at wakeup time. If more than one event has to occur and the order is not known (the two event are asynchronous or unrelated) the best choice is to have more Tasks, one for each operational stream.

The WAKEUP function may be called by any Task to restart the execution of another Task whose TDE address is in R1. This function is used to synchronize tasks, and may be called by the Interrupt Handler Routines. If R1 = 0 then R0 holds the calling Task’s TDE address in return.

Dispatcher

The Dispatcher routine starts with masking the PSW for interrupts, to avoid interference with Interrupt Handlers, then it scans the ECB list for posted ECB’s, for each one found the associated Task Descriptor Element is posted. Then the Task Queue is scanned to locate the first Task with POSTED and WAITING bits in TASKSTAT both set or both zero, if the Task is not disabled (LIMBO bit on) and its priority is zero the search loop is abandoned, the PSW is unmasked, the Task becomes the Running Task, it is put in front of the Task Queue with the RUNNING bit on in TASKSTAT, all its registers are restored from its savearea and a branch to the address contained in R14 is taken. If its priority is not zero, the TDE address is saved in BESTTASK and the search loop is resumed. Unless a zero-priority Task is found later, each executable Task found is compared with BESTTASK, and replaces it if its priority is smaller. If the end of Task Queue is reached the BESTTASK becomes the Running Task. Zero-priority Tasks demand less overhead to the Dispatcher. All Tasks normally run with zero priority.

Interrupt Handling

Interrupts are intercepted by means of the substitution of the New PSW in the low Nucleus of CMS. Attempts to use the CMS macro HNDINT were not successfull because of interaction with CMS macro LINEDIT.

Upon I/O interrupt the Device Table Queue is scanned for a match with device address, if no match a branch to the CMS interrupt handler is taken. If a match is found the CSW is copied to the Device Table and inspected to select the proper ECB (synchronous = Device End, or asynchronous = Attention). If the Task pointer of the ECB (2nd word) is zero there is a branch to CMS interrupt handler (e.g. LINEDIT on the console). If it is non-zero the associated Task’s savearea is located and the CSW is ORed to TASKCSW, and if the Device End or Channel End bit is present the ECB is posted (Device dependent).

The External Interrupt handler scans the External Table queue to find an interrupt code match (for now only VMCF code X‘4001’ is handled), if there is no match the CMS External Interrupt handler is branched to (may be Timer = X‘1004’ or ATTN = X‘0080’), if there is a match the associated ECB is posted.

All interrupt routines run disabled for any interrupt, but for external interrupts additional processing is required which is not possible in the interrupt routine, so the Control Register 0 is used to provide additional masking after the interrupt routine completion, the PSW System Mask alone being unreliable running together with CMS. Each External Interrupt Table has a disabled and an enabled mask to OR or AND to CR0.

The Program Check Interrupt handler, in case of operation exception, looks first if the invalid operation code is the special ABEND code (X‘03’), in this case an ABEND message with the code
contained in the subsequent byte is displayed on the console, otherwise the message 'Program Check XXX at xxxx' is displayed. Then, in any case, all registers are displayed on the console and saved in an area pointed by PROGRES in NETCOM. If there is a Running Task the message 'Task XXXX disabled at xxxx' is displayed with xxxx being the last execution address for that Task and the DISABLED bit is set in its TDE. If there is no Running Task the message 'System Crash' is displayed. The END Task is created using the FIRSTASK entry point, because there is no Running Task now, and the TERM entry is called to shutdown the system. The END task is necessary because the messages are enqueued to MSG task which should run normally to display them on the console, the END Task lowers its priority to 255 and waits for the last message to be displayed before returning control to CMS with the Return Code 2000. In case the Program Check happens during the execution of MSG Task there will be no message display at the Console, and the END Task returns to CMS with Return Code 2002, which means that some undisplayed messages are to be found in the trace of the formatted dump. The same Return Code is used if there is a new Program Check Interrupt during program check processing. After the return to CMS the virtual printer should be closed to obtain the dump output.

ABEND Traps

As explained before operation code X'03' is used for Abnormal END traps, followed by a one-byte code. There is an ABEND macro in NETLIB MACLIB which generates this two byte pseudo instruction, and also generates the referenced symbol ABENDXXX, where XXX is the decimal code, so it is easy to find them in the listings. Each ABEND call has comments which explain the reason. A list of the abend codes used in MSUP follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CRETASK No free storage for new task savearea</td>
</tr>
<tr>
<td>1</td>
<td>CRETASK No free element for new task TDE</td>
</tr>
<tr>
<td>2</td>
<td>KILLTASK Task Descriptor not found</td>
</tr>
<tr>
<td>3</td>
<td>DCLECB No more space in ECB address list</td>
</tr>
<tr>
<td>4</td>
<td>DCLECB New ECB already in use</td>
</tr>
<tr>
<td>5</td>
<td>MSINT CMS External Interrupt Handler changed</td>
</tr>
<tr>
<td>6</td>
<td>DROPECB ECB not found</td>
</tr>
<tr>
<td>7</td>
<td>DROPECB Invalid ECB address list pointers</td>
</tr>
<tr>
<td>8</td>
<td>NETIO ECB not initialized with I/O active</td>
</tr>
<tr>
<td>9</td>
<td>GETBUF No more free storage</td>
</tr>
<tr>
<td>10</td>
<td>MSINT CMS I/O Handler changed</td>
</tr>
<tr>
<td>11</td>
<td>RELBUF Buffer released twice</td>
</tr>
<tr>
<td>11</td>
<td>RELBUF Release a free or invalid Buffer</td>
</tr>
<tr>
<td>13</td>
<td>CRETASK Invalid Entry address or Savearea</td>
</tr>
<tr>
<td>14</td>
<td>EXECTASK Task is already Running Task</td>
</tr>
<tr>
<td>15</td>
<td>EXECTASK Invalid Task savearea address</td>
</tr>
<tr>
<td>16</td>
<td>QUEUE Invalid Task Descriptor address</td>
</tr>
<tr>
<td>17</td>
<td>NETIO Both CE and DE missing</td>
</tr>
</tbody>
</table>

Other modules have ABEND Traps. The best way to identify an ABEND Trap is to use the Program Check Old PSW address, with the LOAD MAP. Then the module can be localized, and the trap found in the Listing file. A comment on the same line of the Trap explains the reason for the ABEND.
Dump

Debugging Aids

Dump

There is a formatted dump utility, invoked as NETDUMP under CMS and with the DUMP command under Network Operating System. It is a transient module (loaded at X’E000’) which issues the CP DUMP command for the pages between 20000 and 30000. It displays Tasks, Device Tables, Link Tables, Spool File Blocks, open CMS File System Control Blocks (FSCBs) and File Status Tables (FSTs). It displays also the last 120 entries of the internal Trace, each one of which consists of a four character identifier and three fullword of arguments. For each identifier a routine is called to decode and display the arguments. On the right side of the trace printout, a segmented line shows the Running Task, each vertical column corresponding to a Taskid. The line is replaced by symbols like ‘*’ for ATTN interrupts, ‘M’ for messages and ‘$’ for START I/O.

Trace

The Trace is in the TRACE module, which consists of a simple routine to fill a circulating buffer, actually the remainder of the memory page. The trace is called with the TRACE or TCINT macros, which save all registers, so they can easily be inserted or taken out from any place in the code. They add some overhead, although not much, and may be deactivated in an entire module defining a dummy macro like this at the beginning of the module:

MACRO
TRACE
MEND

or in the whole system if the TRACE and TCINT macros are replaced by dummy macros in the NETLIB MACLIB.

Log

There is the possibility to log on the virtual printer some information on each transaction for the I/O Devices and VMCF. Only one Device at a time should have the Log on, because there is one only virtual printer. Each device has its own format, the general format is:

XXXXX: Task Name of task issuing the SIO (if first CCW),
Hex display of first 20 bytes of data transferred,
Hex display of the CCW (and of the CSW if last CCW),
Return address, CAW hex display and SIO count (if first CCW).

for VMCF the format is the following:

XXXXX: Task Name of task issuing the Diagnose,
Hex display of first 32 bytes of data transferred,
EBCDIC display of the same area.
MSG

Message Interface

This is the output channel to the operator's console. Any message generated at any point in the System should use the MSG macro, which calls the MSG entry in the MSG Module to encode the message number and up to three parameters into a 32-byte memory element, and queues it to a chain known to MSG Task (normally in wait state). If the message queue was empty the routine calls the WAKEUP function for Task MSG, then, depending on the parameters on the MSG macro call, the MSG routine either waits for the completion of message (the actual display of it - a synchronous message), or just returns to caller (an asynchronous or stacked message), leaving the message queued and resuming Task execution.

A sample of the two ways to use MSG macro is included at the beginning of MSG module in the SAMPLE DSECT. Registers R15 and R1 are used for the stacked form, and R15, R0 and R1 are used for the synchronous form.

At this point the MSG Task takes over, dequeues the 32-byte message elements and searches the message table for a message with that number. The Message Table is a list of 2-fullword elements, the first fullword contains the message editing routine address, the second contains the prototype message text, which will be used by LINEDIT macro to substitute the parameters in the form requested by the editing routine.

When the message is properly edited, a call to the CNSIO entry point in the CONS module is executed to display the message on the console. The actual output of the message is done with a Start I/O, therefore the CMS output translate table is not effective. When the console I/O is completed the MSG Task receives back control and, in case of a synchronous message, calls the WAKEUP function for the Task which had issued the message, releases the 32-byte message element, and then goes back to dequeue another message. When the message queue is emptied, MSG calls WAIT.

The message is displayed in standard form, beginning with the 10-character header NETmmmddd where 'mmm' are the first three characters of the module name (CSECT name), 'ddd' is the decimal message number and 't' is the type of the message, used for filtering. The message types in order of importance are:

- R - Response
- E - Error
- I - Information
- P - Performance
- D - Debug

the default setting for message filtering is I (only I and E messages displayed), however all message types appear in the internal trace, although not all messages may show up, but only the last 64 or whatever the number of the 32-byte elements is. When the memory elements are released, they are enqueued at the end of the free element chain, so they can be still found intact (if not yet used) by the trace/dump system. The formatted Dump module branches to the same routines in the MSG module to decode and edit message elements, this ensures the consistency with the console messages.

Using the 10-character standard dump header the messages are edited according to the CP SET EMSG status, while the formatted dump always shows the complete message. Following the header there is the Time of Day of the message element generation (which may not coincide with the display time, but appears on the internal trace), two blanks and the text of the message. If the message is longer than the terminal line or screen, the rest of it is displayed on the subsequent line aligned with the beginning of the text.

If a message number is unknown the 'Invalid message n. mnnn' error message is displayed.

To add a new message a 2-fullword entry must be added to MSGTABLE at the proper offset of 8*N and a MSGDF macro call must be added at the end of the module. There is no order restriction for MSGDF macros, and there is a limit of 999 for the message number. The MSG module is now just over one page of memory, but there is no limit on the size of the module, only the beginning of the MSGTABLE should be in the first page to be addressable.
CONS-CMD

Command Interface

The module CONS contains the terminal input interface, and the module CMD contains the command interpreter. The task CONS is in charge of processing commands from the console, while the task VCON processes commands received as SMGS. The console input is obtained with a Start I/O, and therefore it does not pass through the CMS console stack and the CMS input translate table. The commands have a set of subcommands and may require some parameters. There is a set of tables, one for commands, and one for each command which has subcommands. The tables consist of several entries containing an 8 character command or subcommand name, a byte representing the minimum abbreviation length-1, 3-byte pointer to a routine. There is only one recursive routine which scans the tables, so that this tree-like structure may be extended to any level (e.g. sub-subcommands, etc.). The scanning routine has a built-in help facility: if 'HELP' or 'H' is typed in place of a command or subcommand the routine displays the entire list of commands or subcommands for that level. This is a list of commands with their subcommands and parameters:

<table>
<thead>
<tr>
<th>Command</th>
<th>Subcommand</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Node(s)</td>
<td>&lt;nodename&gt;</td>
</tr>
<tr>
<td></td>
<td>Link(s)</td>
<td>&lt;nodename&gt;</td>
</tr>
<tr>
<td></td>
<td>Device(s)</td>
<td>&lt;devname&gt;</td>
</tr>
<tr>
<td></td>
<td>Task(s)</td>
<td>&lt;taskname&gt;</td>
</tr>
<tr>
<td></td>
<td>User(s)</td>
<td>&lt;userid&gt;</td>
</tr>
<tr>
<td></td>
<td>File(s)</td>
<td>&lt;spoolid&gt;</td>
</tr>
<tr>
<td></td>
<td>TAG(s)</td>
<td>&lt;spoolid&gt;</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Msglev</td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>Msglev</td>
<td>msgtype</td>
</tr>
<tr>
<td>REStart</td>
<td>Device</td>
<td>devname</td>
</tr>
<tr>
<td></td>
<td>File</td>
<td>spoolid</td>
</tr>
<tr>
<td></td>
<td>Task</td>
<td>taskname</td>
</tr>
<tr>
<td>Please</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG</td>
<td></td>
<td>devname</td>
</tr>
<tr>
<td>CMS</td>
<td>any CMS SUBSET command</td>
<td></td>
</tr>
<tr>
<td>DUMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where capital letters show the minimum abbreviation and angle brackets mean optional parameters.

Commands:

QUERY: it can have several parameters:
QUERY LINKS
QUERY LINK nodename
  gives a list of open Links, or tells if a specified link is still open or has been closed.

QUERY DEVICES
QUERY DEVICE devname
  gives a list of devices known to system, for which a Device Table exists, with the number of Start I/O issued and the number of errors on that device. If a four character Device name is added as a parameter, it displays information only for that Device.

QUERY TASKS
QUERY TASK taskname
  gives a list of all active task (not the queued ones) together with their status. If a Task name is added, only that task is displayed.
QUERY USERS
QUERY USER userid
  displays the list of active users of the VMCF interface (for which a link is open), with their status. If a Userid is added only the information for that user is given.

QUERY START
gives the time and date of the N.O.S. IPL.

QUERY MSGLEV
gives the message level for message filtering, one letter representing the least important message displayed.

SET MSGLEV msgtype|ALL
may be used only to change the message filtering level, 'msgtype' is one letter from E,I,P,D (default is I, any other is treated as Query Msglev), 'msgtype' can also be 'ALL' to be sure that all messages are displayed.

RESTART
it is used to restart a Device, a Spool File, or a Task:

RESTART DEVICE devname
  simulates an ATTN interrupt on the device and sets the RESET bit on the Device Table. On the console it just starts a read (VM READ appears on the screen).

RESTART TASK taskname
  wakes up the task, for instance: RES T CONS provokes a console read, like RES DEV CONS.

PLEASE
just repeats the last line displayed on the console, useful when the screen has been cleared.

LOG devname|VMCF < CLOse >
followed by the Device Name opens the Log on the virtual printer. If 'VMCF' is specified instead of 'devname' the VMCF Header is printed in hexadecimal for each VMCF interrupt or diagnose. When 'CLOse' is added the log is ended, but the printer is not closed.

CMS
followed by any CMS command acceptable as a CMS SUBSET command (i.e. does not disturb the User Area from 20000 hex onward) it executes that command. This feature is very powerful, because adds a large set of commands to the System, e.g. you may edit a file.

DUMP
calls the DUMP transient module to output the formatted dump and NOS internal trace on the virtual printer. The entry point USDUMP is conditionally called.

TRACE
conditionally calls the User Exits TMTRACE and USTRAC. This command is for the User Trace; the NOS internal Trace is printed with the DUMP command.

EXIT
return to CMS, does not wait to close the active links. It should be used only during tests.

HELP
types a list of commands or subcommands available at the terminal. Because the command interpreter is recursive, Help is also a subcommand, e.g. Restart Help gives the list of the RESTART subcommands: Device, File, Task.
VMCF

VMCF Interface

Interrupt handling

This is also the User interface, through which users can communicate with a service machine. VCINIT is the initialization entry point, which creates the VMCF Task. This Task declares to NOS its External Interrupt Table, issues the VMCF Authorize diagnose, and waits for external interrupts with code X'4001'.

Upon interrupt the VMCMUSE field (User Doubleword) is tested for zero, if not zero it is assumed to contain a Linktable address in the first word and is compared to the corresponding doubleword in that Linktable. If there is no mismatch 32 bytes of the VMCF Header are copied to the user's linktable and, if the function is SEND, a dynamic buffer is allocated, a RECEIVE diagnose is issued and the Receiver Task alerted. If the Final Interrupt bit is set, the Sender Task is alerted.

If the User Doubleword is zero or does not match, and the function is IDENTIFY, in the SLAC version a new Link Task is generated, and in the CERN version the linktable chain is scanned for a match of an 8-byte string. When a match is found, if the linktable is multiple, a new linktable is allocated and chained to the multiple linktable chain anchor (VCLKNXLK). If not multiple the same linktable is used. The original multiple linktable is never filled in, it just serves as an anchor for the secondary linktable chain. Every linktable in this chain has a pointer to the anchor linktable (VMCMMULT). At this point the link is open, the user doublewords are exchanged, an acknowledgement is sent with an IDENTIFY Diagnose and the User Exit NEWVMCF is called with R1 = 0.

If the 8-byte parameter does not find a matching linktable, the interrupt is ignored.

The VMCMMDID field is zero for an Open Identify, and non-zero for a Close Identify, where it is the Close Code. The Close operation is to release the Linktable, to destroy the User Doubleword (key) and to call the User Exit NEWVMCF with the Close Code in R1.

If the VMCMFUNC field in the Header is SEND, and the final response or reject bits are set, the sending Task is posted, otherwise, a RECEIVE Diagnose is issued using the buffer specified in the linktable or a dynamically allocated one (this allows greater throughput than waiting for the receiver Task to call VCREAD). If the user has specified no Dynamic allocation of buffer, and the Read entry was not yet called, no action is taken (a slower alternative, but the user can supply his own buffers).

VMCF Entry Points

There are five entry points in the VMCF module, they are:

- VCOPEN to open a link
- VCCLOSE to close a link
- VCREAD to receive a buffer
- VWRITE to send a buffer
- VCLINK to get information about a link.

These entry points are not FORTRAN callable and the registers are not saved, so a suitable interface is necessary. At CERN there is the WINDOWS interface, at SLAC the user interface was merged into the VMCF module. R15 contains zero or the error code on return.

VCOPEN - It is called to open a VMCF Link if R1 = 0, then R2 is the address of the destination Userid, and R3 the address of a two letter string for the characteristics of the Link: Send/Receive/two-way, and Single/Multiple. The second letter can specify also if the buffers are user-supplied or should be dynamically allocated. The new linktable is allocated and filled and the key (user doubleword) is generated in this way: four byte address of the linktable, one byte from the Timer (3rd byte at X'52'), three byte address of the Receive Task, so that, even if two virtual machines close and reopen their link there is no danger of losing track of the linktables.

R10 contains the Task Descriptor Element address of the task to be posted upon Send/Receive
VMCF

completion, or an Asynchronous Exit Routine address to be called. For SENDX and Multiple Links, the running task’s TDE address is used. The linktable is chained LIFO to the primary chain, whose anchor is in the VMCQUEUE pointer in NETCOM. If the link is multiple, the linktable is chained to a secondary chain, whose anchor is in the VCLKNXLK pointer in the multiple linktable. An Open (VMCMMMdd = 0) Identify carrying the doubleword key is sent to the destination virtual machine, the WAIT routine is called, to wait for the acknowledgement Identify carrying the partner’s key.

If R1 contains the Linktable address, this means that positive acknowledgement (ACK) is to be sent (R2 = 0) or a negative acknowledgement (NACK) is to be sent (R2 = close code) in response to an open. An Identify (VMCMMMdd = 0 for ACK, = close code for NACK) with the key is sent to the requesting virtual machine, without waiting.

VCCLOSE - This entry is called to close a Link, R2 holds the close code (zero is translated to 4: normal close), and R9 has the Linktable address. An Identify Diagnose (with VMCMMMdd = close code) is sent to the partner virtual machine, and the Linktable is deallocated from the primary chain or secondary chain if multiple, then the linktable is released (put at the end of the free linktable chain, so that it is not reused immediately).

VCREAD - The read entry is used only when the buffers are supplied by the user, i.e. there is no dynamic allocation of buffers.

R1 is the Buffer address and R2 its length in bytes. The linktable (whose address is in R9) is checked for a Send already arrived. If not, WAIT is called. When the partner’s Send has arrived, a Receive Diagnose is issued, and the user buffer filled.

When the user specifies dynamic buffering in the Open call, there is no read entry to call, so the asynchronous exit routine is called instead, after a dynamic buffer is ready from the VMCF task, which issues the Receive Diagnose.

VCWRITE - The VMCF Write entry point is used for sending data on the Link, it performs a SEND or SENDX Diagnose, depending on the open parameters, and calls WAIT (for the SEND case). When the partner has issued a Receive or Reject Diagnose, and the final response interrupt has arrived, the WAIT is satisfied and a return to the caller is taken.

This operation stops the calling Task for the time the partner virtual machine needs to issue a Receive (an event out of control of the sender virtual machine). A high priority Task which should serve several users in turn will not work correctly because of the serialization of the wait calls. A better solution is to split the one high priority task in several one-user tasks, and take advantage of the parallel operation of different tasks waiting for different events.

VCLINK - This entry is used to retrieve the partner’s linktable address in its own virtual machine (part of the doubleword key) in R0, its name address in R1, and the chain pointers in R7-R8. If R9 is zero instead of the linktable address, the user’s own userid address is given in R1.

Maintenance

At CERN the virtual machine dedicated to the maintenance is NETMAINT which has the following execs:
A EXEC is for assembling NOS modules, it declares the global maclibs and uses the HASM assembler.
TM EXEC assembles the Cernet Transport Manager, the Windows interface, and the CERNET module (Modcomp Driver).
TMLIB EXEC regenerates the TMLIB MACLIB which contains some the Cernet macros (others are accessed on-line from C3.WIE.MACROS on MVS volume CRN102), the Windows macros, and modified MVS macros (EXCPVR, WTO, WAIT and STIMER).
NETG EXEC with the parameter TMG generates the TM module on disk 192 for the CERNET
VMCF

virtual machine, which has the Modcomp front-end attached as 500-501. The TM module must be renamed to TMG after test to go in service.

NETGEN EXEC, called with the user program name as parameter, generates the user module for access to Cernet. Testing programs like TEST and ECO can be generated by NETGEN. This exec is on the P disk.

NETDUMP MODULE is the dump formatter; it produces a formatted dump of memory, tasks, device tables, link tables, FSCBs, and the internal trace on the printer. It is on the P disk.