LBL-14264

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

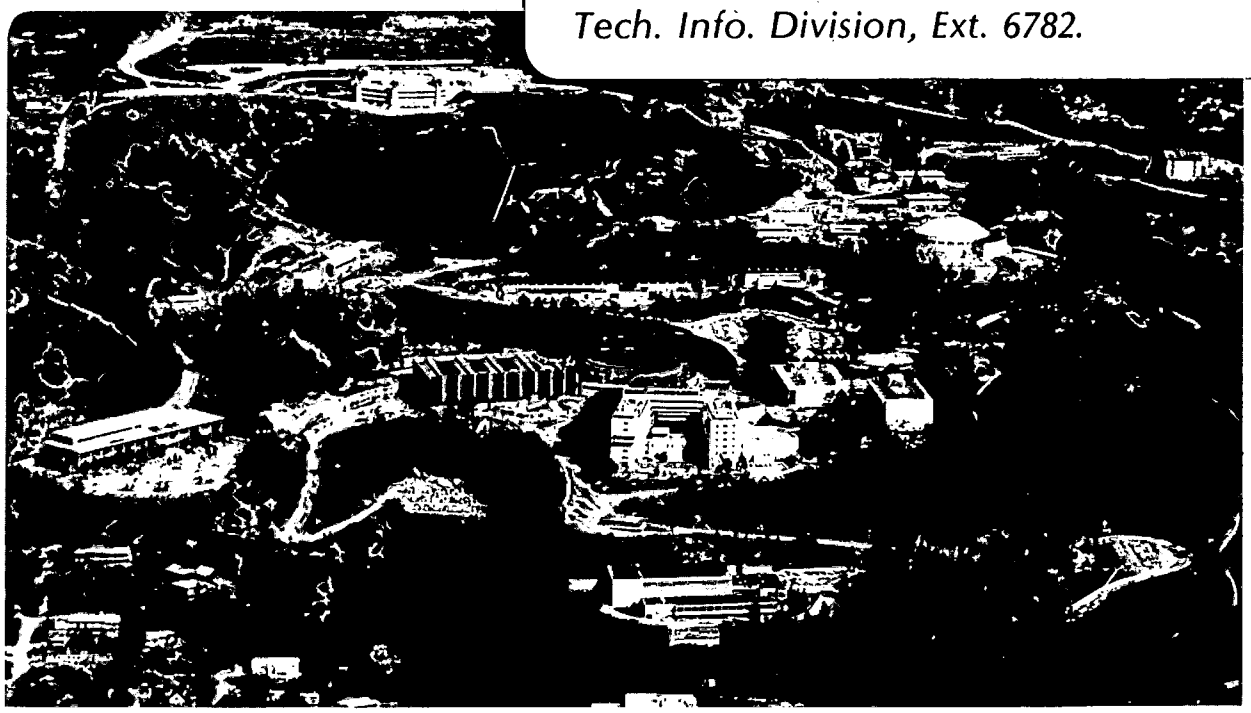
Engineering & Technical
Services Division

Submitted to the SHARE 59 Meeting, New Orleans, LA
August 22-27, 1982

A FORTRAN IMPLEMENTATION OF A NETWORK EXECUTIVE UNDER IBM'S VM/CMS

Martin S. Itzkowitz

March 1982



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

To be submitted to
SHARE 59 Meeting, New Orleans, LA
August 22-27, 1982

LBL-14264

A FORTRAN Implementation of a Network Executive under IBM's VM/CMS*

Martin S. Itzkowitz

Lawrence Berkeley Laboratory
University of California
Berkeley, California

March 1982

---

* This work was supported by the U.S. Department of Energy under contract number DE-AC03-76SF00098.

A FORTRAN Implementation of a Network Executive under IBM's VM/CMS*

Martin S. Itzkowitz

Computation Department
Lawrence Berkeley Laboratory
Berkeley, California 94720

Abstract

The Lawrence Berkeley Laboratory Computation Department has attached an IBM-4331 to a heterogeneous local network. The network, named THC, uses Network Systems Corporation HYPERchannel hardware to implement a general process-to-process communication scheme. It appears to the user as an extension of the local operating system accessed by a subroutine call. The executive multiprocesses requests from different user virtual machines and exchanges messages with other nodes on the network. We describe an assembly-language module, occupying one page of memory, which allows the straightforward CMS-FORTRAN implementation of such an executive as a virtual machine, requiring no modifications whatsoever to the standard operating system. We then describe our network executive as an example of its use.

*This work was supported by the U.S. Department of Energy under contract number DE-AC03-76SF00098.

Introduction

In the Spring of 1980, Lawrence Berkeley Laboratory purchased an IBM 4331 to provide a gateway service to our existing local network. A Network Systems Corporation HYPERchannel adapter, model A220, was attached to a block-multiplexor channel on the machine, and connected to our existing CDC and DEC machines. For our purposes, it is used as two independent devices: one device and subchannel is used for sending messages to other machines; another is used to receive messages from other machines.

The network protocol, named THC, establishes an interprocess communication scheme which requires only three functions: establishing a connection (i.e., virtual circuit), sending data over it, and closing it. Functions are performed in response to requests; a request contains a block of up to 3840 bytes of data, and may return to the user another similar block. The network protocol is described elsewhere[1].

To the user, the network executive appears to be an extension of the local operating system; it must multiprocess requests from user virtual machines, exchange messages with other machines, dynamically allocate buffers and queues, and provide information to a console operator.

In other implementations, modifications to the operating system have been necessary to recognize and process network requests; however, VM provides the necessary tools so that, as suggested in the System

---

[1] Knight, Jeremy and Marty Itzkowitz, THC - A Simple High-Performance Local Network, LBL Report 11426, August, 1980.

Programmer's Guide[2], such an extension may be written as an independent virtual machine interacting with user machines by means of the VMCF interprocess communication protocol. The CMS system allows direct access to dedicated I/O devices, external interrupts, a real-time clock, and, using the diagnose interface to CP, services such as VMCF and paging. These features are all that are needed to implement a network executive, but, unfortunately, they are accessible only to the assembly-language programmer.

CMS supports several higher-level languages, among them FORTRAN. Although usually considered less than ideal for implementing complex systems, FORTRAN does have several advantages. It is compiled into relatively efficient object code, it is familiar to most scientific programmers, and it has a simple interface to assembly language subroutines. We here describe a set of FORTRAN-callable subroutines providing graceful access to the full resources of the virtual machine with no modifications to either CP or CMS.

The package is a single assembly-language module, named BKYEX@, with multiple entry points; it contains handlers for external and I/O interrupts, occupies one page of memory, and is initialized upon the first call to any of its subroutines. It extends the FORTRAN language to allow multiprocessing, to handle external interrupts, to process requests from other virtual machines, to handle a dedicated I/O device, and to dynamically allocate its own memory. We will discuss each of the subroutines in turn, and then describe the network executive as a case

---

[2] IBM VM/SP System Programmer's Guide, publication SC19-6203-0, page 159.

study of their use.

Multiprocessing

One method of multiprocessing is based on a set of event flags, each of which is associated with a subprocess. The main program of such a multiprocessing system waits for an event flag to be set, determines which flag it is, takes the necessary action for that process, and then returns to sleep until another flag is set.

For our system, we used a set of sixty-three event flags. One subroutine, **EXSET(I)**, sets the Ith flag; another, **EXCLR(I)**, clears it. Two functions perform test-and-clear operations on the flags. The first, **IFEXON(I)**, returns zero if the Ith flag is clear, and the value of I if the flag is set. If called with I equal to zero, it will return the value of the highest flag set, or zero if no flag is set. The second, **IWAITX(I)**, behaves similarly, except that it enters an enabled-wait state for the virtual machine rather than return a zero value.

Of course, such a scheme is predicated on the ability to set flags in response to asynchronous events. Subroutines to associate flags with external and I/O interrupts are described below. In order to perform non-interruptible operations, two subroutines, **DISABL(MASK)**, which stores the current interrupt mask and then disables all interrupts, and **ENABLE(MASK)**, which restores interrupts, are supplied.

External Interrupts

Three resources necessary to implement the network executive are accessible through external interrupts: a real-time clock, an operator interrupt mechanism, and the VMCF interface. In order to provide access to these interrupts, upon initialization, the package replaces the External-New-PSW with a pointer to its own routine. (An earlier version used the HNDEXT macro to interface with CMS, but the additional overhead seemed pointless.) Interrupts are enabled for the clock comparator and VMCF, the comparator is set to tick at the next second, and a VMCF authorize function is issued. Since the CMS debug package also uses external interrupts, it cannot be used in conjunction with these routines; CP debug commands are unaffected, however, and were quite adequate.

When the clock comparator ticks, the external interrupt handler updates the real-time clock and issues a diagnose '0C' instruction to update the virtual-elapsed-time and cpu-time clocks. One subroutine, **JSEC(I)**, returns the real-time in Julian seconds, that is, as an absolute integer count of seconds since January 1, 1980. Similar subroutines, **VSEC(I)**, and **CPSEC(I)**, provide integer values for virtual-elapsed-time and cpu-time. Another set of subroutines **JCLOCK(I,J)**, **VCLOCK(I,J)**, and **CCLOCK(I,J)**, associate flag I with an interval timer that ticks every J seconds of their respective clocks. Formatted strings giving the current date and time are returned by subroutines **DATE(ADATE)** and **TIME(ATIME)**.

We allow the operator to interrupt the program by associating an external interrupt I with flag I. The terminal handler in the VM operating system provides a simple command to generate external interrupts; some are used to invoke the formatting of various displays which are then sent to the console; others are used as commands or debugging aids.

Interprocess Communication

The send/receive protocol of VMCF provides a mechanism which is precisely that needed for network requests. The user virtual machine formats the VMCF header in an ten-word array, IVBLOK; a function, **LOCF(ARG)**, which returns the address of its argument, is used to set up pointers. Subroutine **VCSNRC(I,IVBLOK,IRBLOK)** is called to issue the send/receive function. When the response interrupt arrives, its header block is copied into array IRBLOK, and flag I is set. A call to **VCCNCL(I)** will causes the cancellation of the pending request associated with flag I, and, although we do not use it, a corresponding subroutine, **VCSEND(I,IVBLOK,IRBLOK)**, is available for the VMCF send protocol.

Unsolicited, that is, sink-type, interrupts are rejected unless the user has established a means for handling them. A call to **VCAUTH(EXTSUB)** specifies a user-provided subroutine that will be called to process these interrupts. It will be called as EXTSUB(IVBLOK,IREJ) with interrupts disabled; IVBLOK is an array containing the interrupt header, and IREJ is to be set non-zero if the interrupt handler is to reject the transmission. Neither EXTSUB nor any subroutines it may call

should ever be called with interrupts enabled. A debugging aid, subroutine **HANG**, which immediately stops execution of the virtual machine, was used to uncover this reentrancy bug.

Subroutines for sink functions are provided: **VCRCV(IVBLOK)** reads the data corresponding to a send or send/receive call; **VCRPLY(IVBLOK)** replies to a send/receive function; and **VCREJ(IVBLOK)** allows program-controlled rejection of unsolicited messages. One other subroutine, **VCIDNT(IVBLOK)**, sends an identify block to another virtual machine; it is used to announce the restart of a user virtual machine to the network executive. Sendx, resume, quiesce, and unauthorize functions have not been provided: although they are trivial to write, we did not need them.

(An additional routine, **THCTOD(NAME)**, provides a variant of VCSNRC specific to our network FORTRAN interface: it differs in that the response block is unpacked at interrupt time, and the flag is specified by the VMCF message id.)

Dedicated I/O Devices

Before any I/O operations are performed on a dedicated device, the program must establish a means of handling its interrupts. A subroutine, **IOFLAG(IFLAG,IDEV,ICSW)** is used to issue a HNDINT macro for device IDEV, defining it as device "FLii," where "ii" is the hexadecimal equivalent of IFLAG. When an interrupt from the device is received, and the channel status word for the interrupt contains either device-end or unit-check flags, or has a non-zero channel status, the channel status word is stored in ICSW and flag IFLAG is set. Other (intermediate)

interrupts are ignored; a simple change would allow user processing of all interrupts.

I/O operations are initiated by issuing an SIO instruction specifying a channel program for the device. The desired channel program is formatted in an array IPROG, which must be double-word aligned, and subroutine **EXSIO(IDEV,IPROG,ICC,ICSW)** is called to issue the SIO instruction. ICC will be set to the condition code set by the instruction, and ICSW will contain the channel status word. For our purposes, the other I/O instructions were not needed and were not coded; they would be simple variants of the SIO routine.

Dynamic Memory Management

User requests and responses and messages exchanged with other nodes of the network are quite similar. Each has a text, 0-3840 bytes in length, a set of descriptor words giving its mode and length, and control information amounting to some thirty to fifty additional bytes. It seemed reasonable to use a common format for all, and keep them in 4K-byte buffers, aligned on page boundaries.

Memory for N buffers is allocated as a single block of (N+1)*1024 integer words. Upon initialization, the program calls LOCF to obtain the address of the block, and then computes an offset to the first word of the next memory page. This offset is used as a base index for the first buffer; increments of 1024 are added to compute indices for the other buffers. Data is stored in the buffer relative to its base. Allocation and deallocation of buffers is performed with interrupts

disabled. One last subroutine, **DEPAGE(IFWA,LWA)**, which asks CP to release any memory pages between IFWA and LWA, is called whenever a buffer is deallocated.

The Network Executive

The THC executive has been specified in a machine-independent form[3]. It consists of four coroutines: the **request-processor** interfaces between the user and the network; the **listener** processes some incoming messages and dispatches the remainder to the request-processor; the **driver** transmits outgoing messages over the HYPERchannel and passes incoming messages to the listener; and the **housekeeper** is invoked every second to tidy things up. These routines share a pool of buffers and interact through network tables and queues of messages.

The principal tables of the executive are the **node-table**, the **connection-table**, and the **statistics-table**. Virtually all tables and variables are kept as four-byte integer arrays in a single labeled common block; a few statistics are floating-point.

The network executive uses queues of messages and requests. Each queue is associated with a flag and a list head. If the list head is zero, the queue is empty, and the flag is left cleared. If it is non-zero, it is the index of the first buffer on the queue, and the associated flag will be set. One of the buffer control arrays contains the index of the next buffer on the queue, if any. The routines that push

---

[3] Itzkowitz, Martin S. and Jeremy Knight, THC: Design Specifications, LBL report 14144, October, 1981.

buffers onto a queue or pop them off maintain the associated flag.

The network virtual machine is named THC and is autologged by the system. Its profile EXEC spools console output as an error log, bestows some performance-related privileges on itself, dedicates the channel to itself, resets the adapter, and then invokes the network executive. In our configuration, the channel has only the one adapter attached; no coding changes would be necessary to use a shared channel with dedicated devices.

The initialization routine sets up the tables, assigns flags for the console processor, the request-queue (request-processor), a one-second julian timer (housekeeper), the input subchannel of the HYPERchannel adapter (input-driver), the output subchannel (output-driver), the receive-queue (listener), and the send-queue (output-driver, also). The program then exits to its main loop.

The main loop calls IFEXON to determine the highest priority flag set. Console, I/O and the clock flags invoke the call of their respective coroutines. Flags for queues serve merely to wake up the executive: the queues are checked independent of the flag interrupt. When no more flags are set, if the output-driver is idle, the main loop pops the send-queue, and calls the output driver. Then, it pops the request and receive queues calling the request-processor and listener, respectively. When the queues are empty, the main loop calls IWAITX(0) to await further action.

The network virtual machine normally runs disconnected; if necessary, an operator or programmer may connect to it. Some CP external interrupts are used to invoke displays of system tables. Displays are

available giving the status of the whole network, information about the connections currently open or pending, and statistical information about traffic and resources. Other external interrupts are commands to turn any down nodes up, to reset the HYPERchannel adapter, or to reinitialize statistics.

Message traffic between machines is handled by the two subchannels of the HYPERchannel adapter. After initialization, the output subchannel on the device is idle, and the input subchannel is executing a channel program which waits for an incoming message, reads it into a preallocated buffer, and generates a final interrupt when the message has been read. When the input I/O interrupt flag is triggered, the buffer is queued to the listener, another buffer is allocated and a new channel program is begun.

Buffers containing messages to be sent out are queued on the send-queue. If the output driver is idle, and the queue non-empty, the executive pops a buffer from the queue and formats a channel program to send the message. When the completion interrupt is received, the next buffer is sent, and so forth.

The VMCF sink interrupt for an incoming request invokes a subroutine which allocates a buffer, reads the VMCF header into it, and places the buffer on the request queue. When the buffer is popped from the queue, the request-processor issues a receive function. Some requests are completed immediately; others cause the generation of a message to be sent to another machine, and are completed when a return message is received. A request is suspended by attaching its buffer to the appropriate connection-record. The listener, when the message arrives,

or the housekeeper, should the connection time-out, detaches the buffer from the connection-record and requeues it for completion by the request-processor. The text of user requests and replies are used unchanged in the network messages: rather than copy the data, the network executive merely switches pointers. Under some circumstances, a request cannot be processed because no buffer is available; in such cases, the request-processor puts the request on a delay queue which is popped by the housekeeper.

Incoming messages are queued to the listener. Some are replied to by the listener, others are attached to connection-records and later used to complete user requests, and still others are discarded upon arrival.

Conclusions

We have described a set of subroutines that allow the FORTRAN user access to the full resources of a virtual machine. We have shown how these routines allow multiprocessing, external interrupt processing, I/O device handling, and memory management. We have described an implementation of a local network executive, using this package to process requests from other user machines, without requiring any modifications to the underlying operating system.

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720