*Off-line diagnostic programs for system test, acceptance test, and field maintenance of the* IBM 9020 *multiprocessing system are executed under control of the monitor under discussion.*

*The structure of the monitor is based on seven functional components which are examined in detail.*

*A discussion of the debugging experience during the development of the monitor is included.*

# An application-oriented multiprocessing system

# V The diagnostic monitor

**by R. Suda**

The Multiprocessing Diagnostic Monitor (MDM) is a special-purpose control program for the IBM 9020 system. Although MDM can control any problem program written to the proper specifications, the primary developmental goal was to achieve a flexible capability for controlling diagnostic programs in the course of system test, acceptance test, and field maintenance.

Our purpose here is to describe the structure of MDM and the main techniques used in its development. The MDM-controlled diagnostic programs are not discussed. For the most part, the diagnostic library is based on serviceability and diagnostic methods along the lines of those described by Carter, et al., in a SYSTEM/360 context.[1] Because the fault-reporting capabilities of the 9020 equipment extend the standard SYSTEM/360 capabilities (see Part II of this paper), the 9020 diagnostic library is larger, of course, than the standard library. For brevity in the sequel, an MDM-controlled diagnostic program is called a *section*.

The minimum subsystem needed by MDM is a Computing Element, I/O Control Element, Storage Element, Tape Control Unit, and IBM 1052 printer keyboard; additional elements can be controlled up to the maximum system. For the sake of flexibility, which is needed particularly during early system test and field maintenance, MDM permits the operator to define a list of MDM-available elements, and then ensures that only these elements are used or tested by themselves. This is necessary in the field, where it is probable that one subsystem is active on the real-time task and another one is performing maintenance. MDM incorporates vari-

ous options for control of sections; these include options for cycling through sections, for cycling on error, and many others of utility to the field engineer.

MDM permits three modes of operation. The first is a sequential mode in which no section executions are overlapped. The second is a multiprogramming mode in which section loading, section execution, and output printing are overlapped, the intent being to give the elements in the defined system a more thorough and realistic system test and to help in tracking down intermittent failures. In the third MDM mode (which permits multiprocessing), two, three, or four Computing Elements simultaneously execute a series of sections. Since the intent of multiprocessing is to heighten the degree of simultaneity, multiprogramming is always used in the multiprocessing mode. The benefits derived from the multiprocessing mode are similar to those of the multiprogramming mode.

Wherever possible, MDM was written in re-entrant code, viz., in routines that are not modified during execution and hence can be executed by two or more Computing Elements simultaneously. Tables and work areas that are assigned to and can be modified by a given Computing Element are located in a preferential-storage area. Common tables, which are open to modification by more than one Computing Element, are controlled by lock bytes; in conjunction with the TEST AND SET instruction, a lock byte prevents access by more than one Computing Element at a time.

The extensive error-checking facilities of the 9020 equipment are complemented by an MDM error-handling module that gathers and prints information when an error occurs. For most types of errors (particularly for those that cause machine-check interrupts), recovery is not attempted; the program has accomplished its purpose once an error is detected and diagnostic information printed.

Since some of the diagnostic sections require the use of specific Storage Elements, MDM was given the ability to dynamically relocate itself (including re-entrant code as well as preferential-storage areas). When a section makes a request that causes MDM to relocate itself, MDM idles all Computing Elements, relocates itself, and resumes Computing Element operation at the point of interruption. MDM can be interrupted at any point and resumed at that point in a new location in main storage.

### MDM components

The main functional modules of the Multiprocessing Diagnostic Monitor are:

- Initialization
- Input message analysis
- Storage allocation
- Scheduler
- I/O initiation
- Interrupt handling
- Output message assembler

initialization

The initialization module is designed to start sections in as many as four Computing Elements. There are two phases in the initialization process. The first occurs after IPL (initial program load) and initializes a system containing a single Computing Element; the second, which follows certain operator input messages, initializes additional Computing Elements. In the first phase, the several actions completed before operator intervention include seting crucial registers (base registers, program status words, configuration control registers, address translation registers, preferential-storage base address registers, and the select register), clearing logout areas, and establishing the IPL system elements in MDM tables. The IPL elements, which comprise a minimum system, include a Computing Element, I/O Control Element, Storage Element, and loader control unit—usually a Tape Control Unit. After being enabled for I/O and external interrupts, MDM is ready for an input message from a 1052 printer keyboard or card reader. At an attention interrupt, MDM assigns the 1052 as the input device and primary output device. Through a console interrupt, MDM can define the card reader as the input device.

The second initialization phase is triggered by input message. As each additional Computing Element is brought into the system, a preferential-storage area is established and its configuration control register and address translation register are set. Then a write-direct external start signal is sent to the Computing Element, thereby setting the program status word (PSW) from the first eight bytes of the configured Storage Element with the lowest address. Initialization is then completed by the just-started Computing Element. Looking for a task to perform, the Computing Element then branches to the scheduler module.

input message analysis

The Input Message Analysis Module handles the decoding and interpretation of input messages. Typical messages provide information needed by MDM while loading and running sections, changing equipment configurations, and setting monitor options. All messages are first edited by a common routine that handles deletions, blanks, etc. Control is then passed to individual routines, as indicated by the verb in the message (e.g., load, run). The individual routines check for format errors and, if errors are found, return appropriate messages to the operator.

Some of the actions requested by input message are stacked and processed by other MDM modules in normal sequence. For example, a load message is passed on to the loader module. Other requested actions may lead to an immediate transition in execution, as in the case of messages that require a change in job sequence or equipment configuration.

storage allocation

The Storage Allocation Module controls the assignment of main storage with blocks of two sizes. In loading programs and setting up work areas requested by programs, blocks of 4096 bytes are used. Blocks of 128 bytes are assigned by MDM for output and load messages. This module accommodates the maximum storage allowed by the system (twelve 128K-byte Storage Elements).

Each Storage Element is divided into 32 large blocks; a one-bit indicator is maintained in MDM for each block in the system. Thus, it takes one word (32 bits) to control large-block allocation for each Storage Element and twelve words to control a maximum of Storage Elements. Requests for multiple contiguous large blocks can be honored by MDM.

The 32 small blocks in each large block are chained together such that the last block in the chain serves as a control area. If all 32 blocks are allocated and another request is made, a large block is obtained, subdivided, and its control block is chained to the preceding control block. Only one small block can be requested at a time. As small blocks are returned, a check is made to see if a large block can be returned to the large block pool. In general, large-block requests from the small-block routine are served with blocks from upper storage and similar requests for section loading from the lower end of storage. This tends to leave the middle of main storage free to satisfy requests for multiple contiguous blocks.

Two of the six macroinstructions associated with main-storage allocation, GET LARGE BLOCKS and RETURN LARGE BLOCKS, allow requests for contiguous blocks. Two others, GET SMALL BLOCKS and RETURN SMALL BLOCKS, serve requests for one block only. These four macroinstructions are self-explanatory. The last two, GET DEFINED AREA and RETURN DEFINED AREA, permit requests for one or more contiguous large blocks with a specific storage address. If the requested area is not available, the requesting program is delayed until the desired area is released. If the requested area contains MDM, the preferential-storage area, or the requesting program, then MDM relocates the contents of the defined area to complete the request.

scheduler

Two schedulers exist within MDM, the Task Scheduler and the Section Scheduler. An MDM "task" is work that requires an MDM module. The MDM Task Scheduler has priority to complete all MDM tasks that can be run before entering the Section Scheduler. The Task Scheduler scans the task table (which contains one entry for each MDM module) for work to do. Work called for by a given entry is queued to that entry and identified in a small storage block.

The Section Scheduler is entered after all MDM tasks are completed. It scans a program-status table, which contains one entry for each section in the system, for sections to be run. An entry contains information the scheduler needs about a section, such as starting address, PSW setting, and section options. In sequential mode, each section is run to completion and no monitor I/O, such as loading, is overlapped with section execution. In multiprogramming or multiprocessing mode, each time the monitor gains control, it steps to the next entry in the program-status table. Thus, each section being multiprogrammed gets a somewhat random share of running time, a circumstance that is often desirable in a diagnostic system.

Once each pass through the schedulers, I/O and external interrupts are allowed, so that new tasks can enter the system and

interrupts from any pending sections can be processed. At other times, MDM is run with interrupts disabled, whereas sections are normally run with interrupts enabled.

I/O initiation

The I/O Initiation Module is concerned with I/O for both MDM and I/O section use. Within MDM itself, there are two primary I/O needs, section loading and message output. The section loader employs a Load Message Table which contains, among other things, the identities of sections to be loaded. The entries in the table are set upon receipt of a load message from the operator or upon request from a section via an SVC (SYSTEM/360 supervisor-call instruction) interrupt. Sections appear on a library tape with their identities in collating sequence. There are three records per section—the header, text, and relocation records. To load a section, the library tape is searched on header records until the desired record is found, storage is requested, and the text and relocation records are read. The relocation record contains relocation data that permit respecification of address constants in the text. If the section requires I/O devices, these are assigned next. If the requested I/O equipment is not assigned, the section run is delayed until the equipment becomes available. A storage protection key is set for the section, and an entry is completed in the first open cell of the program-status table. The section is then ready to run.

The message-output portion of I/O initiation is responsible for messages from both MDM and the sections (message formatting is governed by another module); it directs device operations and checks for I/O error conditions. Another portion of the I/O Initiation Module is normally responsible for the main I/O operations of sections. (However, a section can choose to run in the supervisor state and thereby take responsibility for initiating all operations and handling the resulting interrupts.) This portion gains control from the Interrupt Handling Module (described later) as a result of an I/O operation for a section running in the problem state. A check is made to see if the device specified in the I/O operation has been assigned to the section. If the path to the device is not busy, the I/O operation is executed. However, if the path is busy, the I/O operation is queued for later execution, and control returns to the MDM scheduler.

interrupt handling

MDM has one interrupt-processing routine for each of the five types of 9020 interrupts. Initial handling, such as saving general-purpose registers, is done before control is passed to one of the five interrupt routines. Optionally, a section can be set up to handle its own interrupts, but they are typically handled by MDM.

*Machine-check interrupt.* If a machine check occurs during the operation of a section, a test is made to see if the section wants this type of interrupt returned. If yes, a return is made to the section via its machine-check PSW. If no, the logout occasioned by the machine check is printed and the section aborted. If the error occurred during the operation of MDM, the logout is printed, the run is terminated, and MDM must be reloaded.

I/O *interrupt.* There are three classes to consider: MDM device, section device, and unassigned device interrupts. At normal-completion interrupt from an MDM device, control passes to the proper MDM module (I/O Initiation or Input Message Analyzer). Channel and control unit completion interrupts from MDM devices are queued until device completion is received. If a unit check occurs on an MDM input device, a repeat message is printed.

Whenever an I/O interrupt occurs in a section enabled for I/O interrupts, return is made to the section via its new I/O PSW. If the section is disabled for I/O interrupts, the interrupt is queued until the section is again enabled. If an I/O interrupt occurs that does not belong to MDM or a section, an error message is printed and processing continues.

*External interrupts.* Two types of external interrupts are automatically handled by MDM: (1) the READ/WRITE DIRECT instructions if used when MDM is starting or stopping processing in another Computing Element during initialization, and (2) the Console Interrupt Key used in signalling to MDM to read an input message from the card reader when the reader is used as the input device. At other times, these and all other external interrupts can be used by the sections. Diagnose-accessible register interrupts, when not used by a section, indicate that a malfunction has occurred; these interrupts are handled in MDM by printing an error message.

External interrupts assigned to a section cause a return to the section if it is enabled or cause the interrupt to be queued if it is temporarily disabled for those interrupts. External interrupts which do not belong to MDM or a section cause the printing of an error message.

*Program interrupts.* Two classes of program interrupts are considered—those that occur in MDM and those that occur in a section. Because a program interrupt in MDM is viewed as an error condition, a non-recoverable wait state is entered; MDM must be reloaded to continue. Program interrupts in a section are subclassified into four categories by MDM. Privileged-operation interrupts (other than I/O), such as SET SYSTEM MASK or LOAD PSW, cause MDM to simulate the privileged operation and return to the section at the next instruction. In the case of LOAD PSW, return is made to the address in the new PSW. At program interrupt on an I/O instruction, MDM transfers control to its I/O Initiation Module. Other program interrupts are returned to the section if it has so requested. If return is not requested by the section, other program interrupts are considered errors, an error message is printed, and the routine is aborted.

*Supervisor-call interrupts.* SVC interrupts are the basic method by which the section requests MDM services such as terminate section, print message, or load new program. Normally, MDM returns control to the section. In the case of a "section end" SVC, control passes to the next section. A section can always request a return of control after an SVC interrupt.

output message assembler

Various types of output-message requests can be made by a section via an SVC interrupt, the distinctions usually being one of format. This module's responsibility is merely to format the data passed to it by a section and then pass control to the output portion of the I/O Initiation Module.

## Debugging experience

The Multiprocessing Diagnostic Monitor was debugged on the first 9020 system. Since equipment and programs were tested at the same time, the technique found most useful was simply to run in a "hard stop" mode that stopped the machine immediately upon detection of an equipment error. This facilitated tracing whether errors were in equipment or program.

Individual MDM modules were debugged in conventional ways, largely by simulating inputs and checking outputs. The first package of modules put together enabled MDM to cycle itself and respond to input messages, no program loading or running function having yet been attempted. A useful debugging tool at this point was a built-in trace. As each module was run, it would record that fact for later investigation. Built into each module were validity checks on data transfers and table references. An error at one of these checks could be combined with the trace to help in pinpointing problems. The simplex MDM was completed by gradually adding other modules, the various options being checked last.

Duplex debugging presented a more novel situation. The most common single problem stemmed from failures to properly use lock indicators in modifying or referencing tables, as a result of which two Computing Elements would operate on the same table at the same time. Because the symptom usually was a loop of some sort in MDM, these problems were difficult to resolve. It was usually fruitless to trace through the loop since the damage was done before the loop started. The best method seemed to depend upon a close examination of tables for some impossible or improbable condition. With knowledge of the entire program in mind, such examinations would finally lead to the error. It appears that special attention to lockout during the planning and coding phases of multiprocessing programs may repay itself several times over. A lockout failure can be difficult to detect since it may take hours before the conditions that create the failure again occur. On the other hand, MDM code associated with multiprocessing control functions, such as starting or stopping other Computing Elements, turned out to be fairly straightforward to debug.

The step from debugging MDM in duplex to debugging in triplex turned out to be an easy one because all the problems had been solved during duplex debugging. While a configuration with four Computing Elements has not been built or tested, it is felt that little or no special MDM program debugging would be needed for this step.

## Summary

The Multiprocessing Diagnostic Monitor was a successful vehicle for testing the 9020 system at various levels of assembly and integration. It has also been used successfully in acceptance testing. Because its use as a field maintenance tool is still under evaluation, along with many other tools, it is difficult to assess its role in the field environment. It is being used daily at the sites, a continuing improvement program is under way, and no extensive changes have been necessary.

CITED REFERENCE

1. W. C. Carter, H. C. Montgomery, R. J. Preiss, and H. J. Reinheimer, "Design of serviceability features for the IBM SYSTEM/360," *IBM Journal of Research and Development* **8**, No. 2, 115–125 (April 1964).