Discussed is an approach to simulating direct-access devices. An experimental simulator provides the capability to test newly developed I/O supervisors and to test code for new or proposed devices without the benefit of the actual device.

This functional simulator performs all of the search, data-movement, and status-reporting functions of the device transparently to the user. Data-driven (table) techniques enable the user to simulate a number of direct-access devices (one at a time) and measure their would-be performance. Interactive options of the simulator enable the user to check for errors and test the various error routines.

Direct-access device simulation

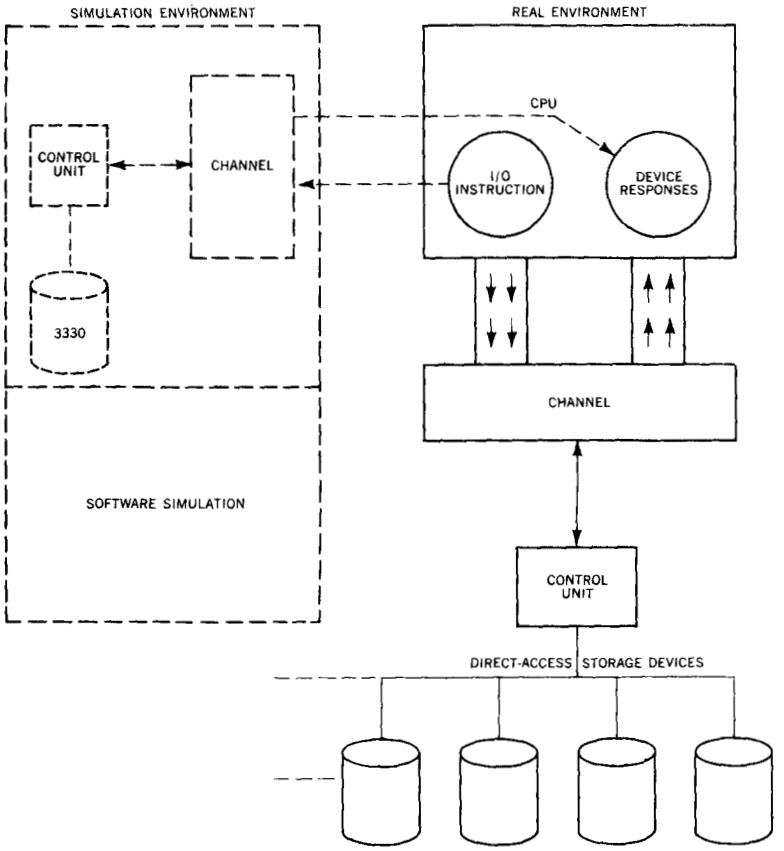
by E. Nahouraii

The term "simulation" refers to a variety of concepts in data processing. Consequently, we need to define its meaning in the context of this paper, that is, with reference to device simulation. In this application, it is the process of reproducing the functions of a physical device, without the device being attached to the system or being replaced by another real device of similar characteristics.

Currently, there are special-purpose device simulators that perform these functions; however, they require special hardware and are thus limited to simulating the one device. Some simulators either do not have data-handling capability or can only be executed on a particular system.[1] Other simulators provide modeling and performance evaluation[2] but no data-handling functions. None of these simulators has the capability of combining functional ability and performance measurements to achieve software testing and to serve as an experimental tool for evaluating hardware and software interactions.

Discussed in this paper are techniques developed and implemented in an experimental approach that enables a user to simulate all channel commands commonly used to support direct-access devices on the IBM System/360 and System/370. No modification of a user program is required prior to execution, thus resulting in a transparent simulator. Data-driven techniques,

Figure 1 Real versus simulated environment

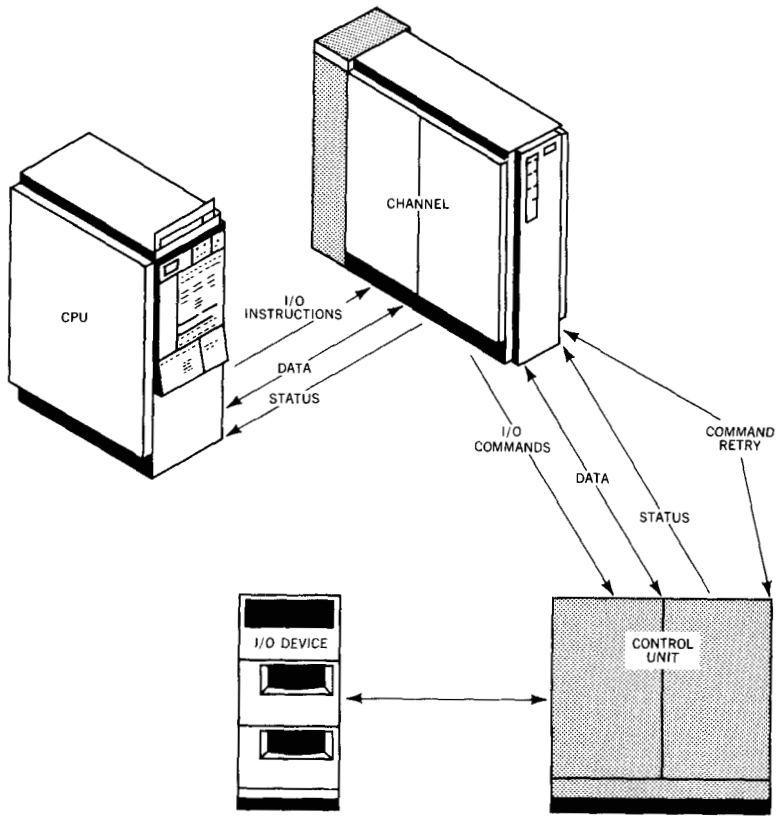


which enable the user to direct the simulation of a number of different devices, are also described. All error conditions, via status, are reflected in the same manner as if the physical device were attached. Figure 1 indicates the setup.

Simulation description

The transfer of information between CPU storage and an I/O device is accomplished by execution of I/O instructions as shown in Figure 2. The direct-access device simulator[3,4] (DSIM) is a software package that intercepts an I/O instruction issued for a device to be simulated. Each I/O instruction is interpreted, and its hardware function is reproduced by testing and modifying data in tables. The tables represent the hardware implied by the combined specification of the channel, control unit, and device. Six I/O instructions perform in the IBM System/370 as follows:[5] "Start I/O" (SIO) identifies the channel, subchannel, and I/O device. "Start I/O Fast Release" (SIOF) causes the operation to be initiated independently of the device. "Halt I/O"

Figure 2 Information transfer between CPU and I/O device



(HIO) terminates execution of a particular I/O operation. "Test Channel" (TCH) determines the condition of the channel. "Halt Device" (HDV) is similar to HIO except that, when a channel is busy, only the addressed device is affected. "Test I/O" (TIO) determines the state of the channel, subchannel, and the device.

When a user executes his application program and tries to write or read data from a direct-access device, control is normally given to the device-level I/O supervisor[6] (IOS). The IOS issues I/O instructions directed to the specified channel, subchannel, and device. DSIM receives control whenever any of these I/O instructions are issued to the particular device to be simulated. Associated with each I/O instruction is a set of channel command words (CCWs) that indicate what specific functions the channel,[7] subchannel, and the device should perform. In the System/370, a fixed location contains the address of these CCWs.

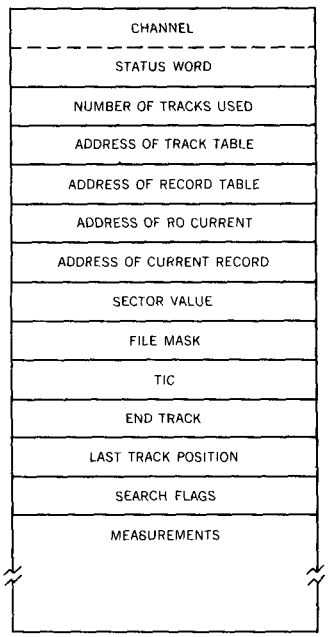
There are five classes of commands: control, search, read, sense, and write. Varous combinations of these commands are

issued by IOS to read or write a record on the device, i.e.,

SEEK
SET FILE MASK
SET SECTOR
SEARCH IDENTIFIER EQUAL
TRANSFER IN CHANNEL
WRITE COUNT KEY AND DATA

When DSIM receives control as the result of IOS issuing any of these I/O instructions, the command associated with the I/O instruction is fetched by DSIM. Each command is simulated independently in the same sequence that the channel, the control unit, and the device would have processed it if they were present. Simulated track size is the same size as the track on the physical device. The detailed simulation techniques can be found in Reference 8.

Figure 3 Device status table



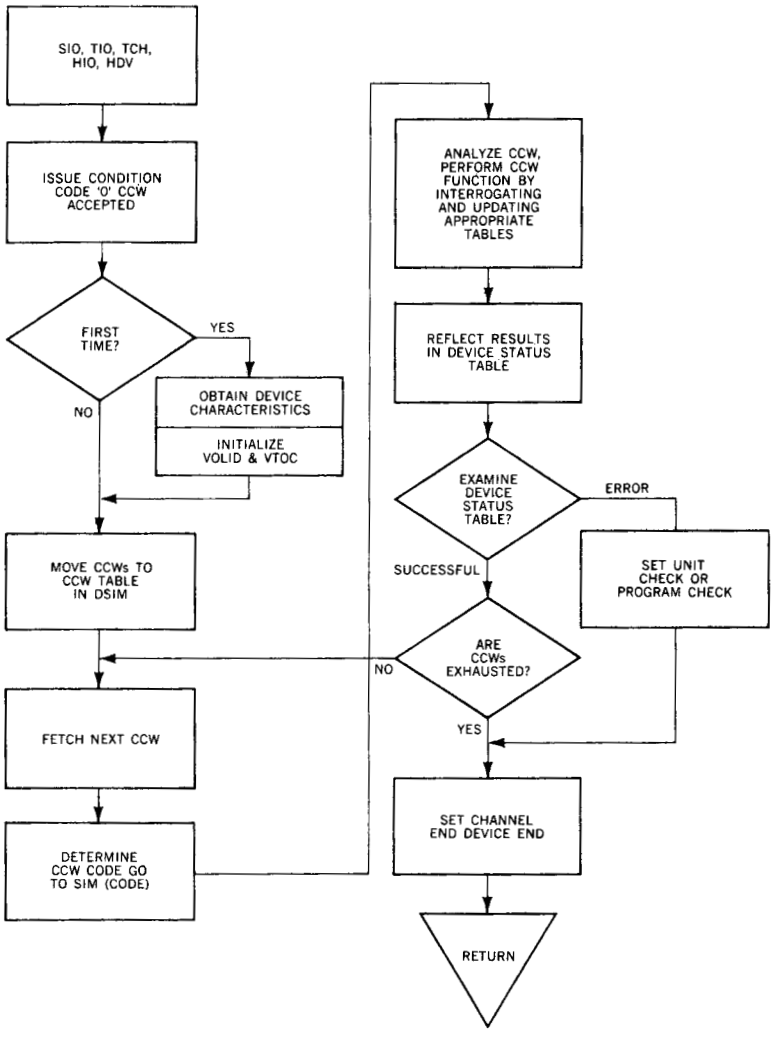
The simulation of a CCW first validates the bit configuration for conformance to device specifications. Following are some of the checks made to effect the validation:

- Format of the CCW to make sure the operation code, address, flags, and count field exist.
- Limits of the data address, data length and count field.
- Valid command code.

All the error conditions resulting from this validation are presented to the system by providing the proper condition code and causing an interruption in the CPU. After a CCW is validated, its code is tested and appropriate subroutines are called to perform the function by interrogating and updating various tables. One such table is the device status table shown in Figure 3, which indicates the status of the simulated device as of the last CCW simulated. The table is updated for each simulated CCW and contains the total tracks that have been used and the position of the read/write heads. When one SIO is completed, the position of the read/write heads is saved before the status of the device is reset. This allows DSIM to calculate the seek distance between the current and subsequent tracks for measurement purposes.

The channel status word table entry indicates the result of the last CCW simulated. Prior to fetching the next CCW, this entry is tested for a channel-end and device-end condition produced by DSIM as illustrated in Figure 4. If these two conditions are true, and the unit check and/or program check are not true (meaning no error has occurred), DSIM proceeds to determine if any more CCWs are to be simulated. If not, the control is returned to IOS by causing a CPU interruption. This interruption is the same as if the real device were present.

Figure 4 Simulation flow



DSIM capabilities

DSIM provides for execution of code using a direct-access device without the device being present. Such a capability is useful during the development cycle of an IOS, especially when the new code must be tested against a prototype device, which often is not available or not fully operational. Without such a test vehicle, it is difficult to differentiate between hardware and software errors. However, with the aid of the DSIM, a developer can test most of his code before using the real device. The DSIM CCW validation is a first step toward testing a new IOS. In addition,

DSIM simulates each CCW and provides indications when the code has generated an unacceptable CCW. The DSIM error-generating mode enables the system to create fictitious errors that permit the testing of error-recovery routines.

DSIM is useful even when the device is available but is at a different geographical location or on a system whose configuration is otherwise unsuitable. For example, it would be possible for application programmers to execute and test their programs before attainment of the real device. Errors can be forced by using the error-generating facility of DSIM. This provides a capability not generally attainable even with the real device.

In general, DSIM is an experimental tool for designers who hypothesize a new device and wish to determine its performance using a real application.

data-driven techniques

DSIM was designed to be a data-driven simulator; thus it allows the simulation of a number of direct-access devices. When DSIM receives control, it will ask the user to type the name of the device to be simulated, for example, an IBM 2314 or 3330 or 3340, etc. If the device type is known to DSIM (one of the above), it automatically initializes the device characteristic tables. Otherwise the user is requested to type the device characteristics consisting of track size, inter-record gap, various timings concerning the read/write head movements, and the data rate. This capability enables a user to experiment with a device by providing different characteristics and then execute an application program to obtain the performance measurements.

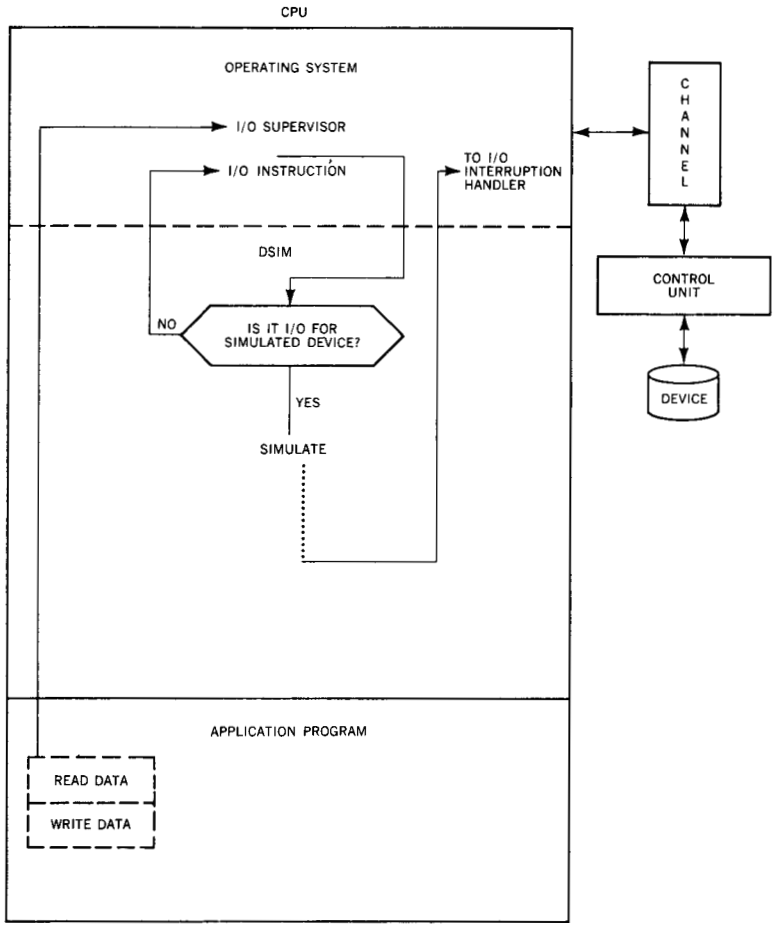
Operating environment

DSIM is designed to operate in either of two environments—the stand-alone machine environment and the multivirtual machine environment. Both modes of operation use all of the same modules, except for one that provides the interface with the host.

stand-alone environment

When the environment is that of a stand-alone machine, the operating system, the DSIM, and the application program reside in the same machine. The operating system is generated to include a channel, a control unit, and a device to be used by the simulator, or a dedicated channel and device can be used. After the system is initialized, modifications are made to the IOS to enable trapping of all I/O operations. DSIM determines if the device being addressed is the one being simulated. If not, control is returned to IOS. Otherwise, DSIM simulates each and every CCW issued. Data produced by the application program and thought to have been written on the (simulated) device actually resides within the machine memory. The application program in this

Figure 5 DSIM in stand-alone environment



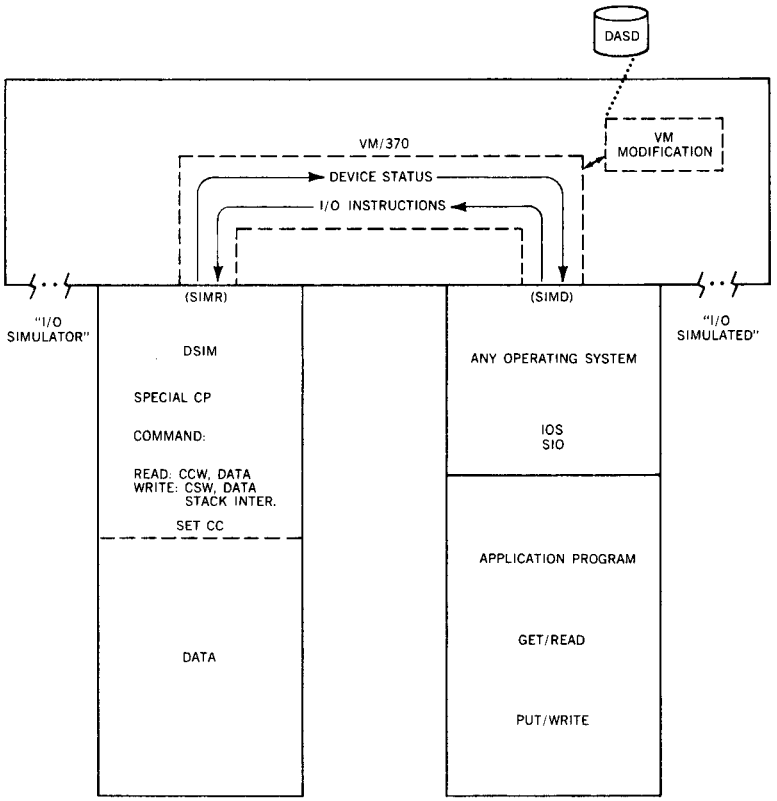
mode can create and update the data as though it were a data set on the device. Modification to the operating system makes this mode of operation release-dependent.

DSIM in a stand-alone mode currently can only simulate IBM 2314 and 3330 devices because the modifications were only made to IBM Operating System/360 release 21. Figure 5 illustrates the machine configuration for DSIM in the stand-alone environment.

multivirtual environment

The IBM Virtual Machine Facility/370 (VM/370) is a system[9] that manages the resources of a single System/370 computer so that multiple computing systems appear to exist within it. Each such computing system is said to run a virtual machine. VM/370 consists of two major components, a control program (CP) and the Conversational Monitor System (CMS) that runs in a virtual machine under control of CP. The multivirtual environment

Figure 6 DSIM in MVE



(MVE) mode of DSIM operates within VM/370 enabling the user to run under any operating system on his virtual machine. Modifications have been made to VM/370 to permit DSIM to obtain the I/O instructions issued for the device to be simulated.

The technique used to provide the DSIM capability requires the definition of two virtual machines. The first is called the input-output simulated (SIMD) machine. SIMD is the machine that contains the user's operating system and application program from which the I/O operations for the device are issued. The second is called the input-output simulator (SIMR) containing the DSIM modules. The SIMR simulates the function of the channel, control unit, and the device and returns all the responses to the SIMD, via CP, as if the real device were present. All necessary data movements, as a consequence of read or write commands, from SIMD to SIMR, or the converse, are also accomplished via this special modification to CP. The SIMD and SIMR machines can be started by using the VM/370 logging procedures.

The user of this support is responsible for executing the operating system and DSIM in the proper virtual machine. He must

define each virtual machine with the appropriate I/O configuration for the simulation and inform CP which two machines are involved. The I/O configuration is defined by using two special commands provided for this purpose. The first command, DEFINE, allows the user to dynamically add a device to his virtual machine configuration, i.e.,

DEFINE SIMD <AS> NNN

NNN is the device to be simulated and consists of a virtual channel, control unit, and device address. The user also must "DEFINE" another device for SIMR, i.e.,

DEFINE SIMR <AS> MMM

MMM is the SIMR virtual channel, control unit, and device address. The second command, COUPLE, can be issued from either SIMD or SIMR, i.e.,

COUPLE NNN TO SIMR MMM

Once the two virtual machines are ready, the application program in SIMD and DSIM in SIMR can be started. Figure 6 shows the SIMD and SIMR in MVE. Other requirements of the user in SIMR depend on the type of options selected. They are described in the following section.

potential capability

The MVE mode of DSIM has other potentials in addition to simulating a number of direct-access storage devices. It is a general-purpose tool that may be used to intercept references and subsequently simulate any I/O device, i.e., printers, terminals, display units, etc.

The simulated device (SIMD) is not restricted to a singular case; a user may define as many SIMDs as necessary. The intervirtual communication capability of DSIM enables the communication of many SIMDs with each SIMR, as illustrated in Figure 7. The SIMD issues an SIO instruction to cause an interruption in SIMR. SIMR has the necessary instructions to move data, messages, or jobs from one SIMD to any other or to/from an SIMR. Figure 8 indicates the commands necessary to establish and bind the SIMR to the SIMDs. This use of DSIM is suitable for experimental networks or data base systems, since it provides send, receive, and store and forward capabilities.

Interactive options

DSIM operates in the interactive mode and communicates with the user to determine what options are to be enabled. Also, if

Figure 7 DSIM intervirtual machine communication capability

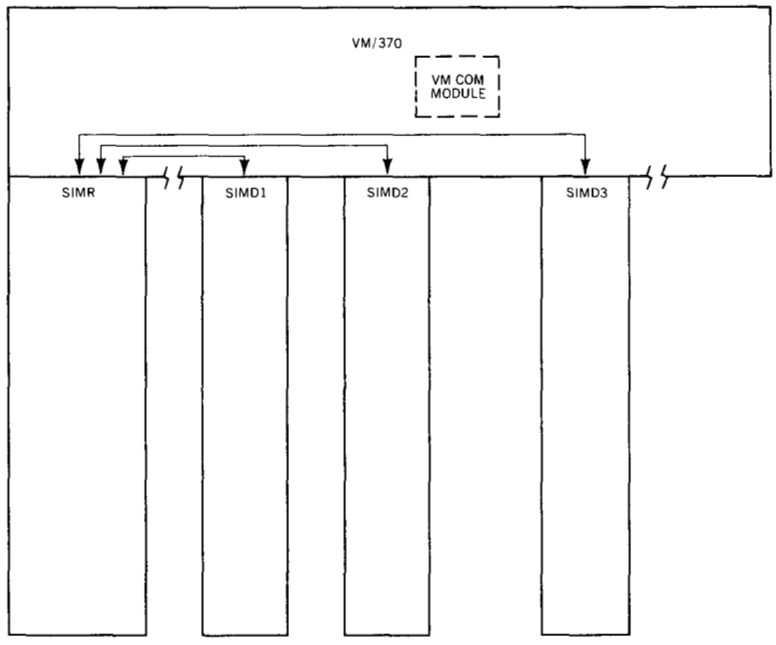


Figure 8 Examples of DEFINE and COUPLE to establish intervirtual machine communication

		SIMR DEFINE		
DEFINE	SIMR	⟨AS⟩	CCU1	
DEFINE	SIMR	⟨AS⟩	CCU2	
.				
.				
DEFINE	SIMR	⟨AS⟩	CCUn	
		SIMD DEFINE		
DEFINE	SIMD1	⟨AS⟩	CCU1	
DEFINE	SIMD2	⟨AS⟩	CCU2	
.				
.				
DEFINE	SIMDn	⟨AS⟩	CCUn	
		Coupling issued either by SIMD or SIMR		
COUPLE	CCU1	TO	CCU2	SIMR
COUPLE	CCU2	TO	CCU2	SIMR
.				
.				
COUPLE	CCUn	TO	CCUn	SIMR

the user is simulating a device not known to DSIM, i.e., not an IBM 2314, 3330, or 3340 direct-access device, it prompts the user to input device characteristics. There are three (optional) modes of operation available to the user—message, error-gener-

Figure 9 Example of SIMR logging

```
vm/370 online                          ljh305  asyesu

       (NOTE:  UPPER CASE=OUTPUT, LOWER CASE=INPUT)


login nahourai m
ENTER PASSWORD:


cp def simr 111
R;

dsim
R;

start mvmintf
SIMR 111 COUPLE BY SIMD          150
EXECUTION BEGINS. . .
TYPE THE SIMR DEVICE ADDR (3 DIGITS).

111
PLEASE GIVE DEVICE TYPE:2314/3330/3340
3330
IF ERROR GENERATING MODE IS DESIRED TYPE YES.

   OR HIT CARRIAGE RETURN FOR NORMAL EXECUTION

PLEASE TYPE YES FOR MESSAGE MODE OR HIT RETURN FOR NONMESSAGE MODE
yes

 TO DISABLE DEBUG MODE TYPE DIS--TO ENABLE HIT CR


 TYPE SIO NUMBER WHERE MESSAGES SHOULD BEGIN, TYPE 1
        TO PRINT THE FIRST CHANNEL PROGRAM
1
```

Figure 10 Message mode

```
SIO          1
      CCW  COMMAND
        1  SEEK
SEEK TK =00000000000000000000000000000000
        2  SET FILE MASK
        3  TIC
        4  SEARCH ID =
COMPARE  0006A577 000126D0 00000005
ID NOT FOUND
        5  TIC
        4  SEARCH ID =
COMPARE  0006A577 000126D0 00000005
ID NOT FOUND
        5  TIC
        4  SEARCH ID =
COMPARE  0006A577 000126D0 00000005
        6  READ DATA
```

ating, and debug. Figure 9 is an example of SIMR logging with a sample of each option as it occurred in the real simulation of a device.

message option

Selection of the optional message mode results in the printing of messages at the console describing the commands as they are being simulated. A count is maintained indicating the number of times each command is executed. It appears in the format of Figure 10.

Another capability provided in the message option is SKIP, which enables the user to specify when these messages should start printing on the terminal, i.e.,

```
TYPE SIO NUMBER WHERE MESSAGES SHOULD BEGIN, TYPE 1
TO PRINT THE FIRST CHANNEL PROGRAM
1
```

error-generating option

When the error-generating mode is selected, the following message is issued before simulation of every channel command.

```
TYPE 0 FOR NORMAL EXECUTION / 1 FOR PROGRAM CHECK /
2 FOR UNIT CHECK / 3 FOR UC & PC.
```

If 1, 2, 3 is selected, DSIM produces a channel-condition device-end condition and the selected error. These errors are also reflected in the SIMD channel status word. This mode of operation assists the system programmer to test various error-recovery procedures and assists the application programmer in testing I/O-related error conditions in his program. This option can be extended, via programming, to produce any desired error conditions that the user wishes to produce.

debug option

By selecting the debug mode, the user receives the addresses of various DSIM tables that are valuable in verifying the correctness of the processing or assisting in isolating errors when they occur. Figure 11 contains examples of these type-outs.

The CCW table contains the CCWs issued by IOS. DSIM, after printing these addresses, halts so that the user can display the contents of the tables.

Restrictions

DSIM simulates the user device in the SIMR machine as long as the operating system or the application program does not include any timing dependencies or any dynamically modified channel programs. Dynamically modified channel programs are those that are changed between the time the SIO instruction is issued and the end of I/O operations; i.e., changed by the channel program or the CPU. The record-overflow feature, which is a means of processing logical records that span track boundaries within a cylinder, is not supported. Not all hardware errors are now simulated, although appropriate software could be written to accomplish the pretense of an error of one's choice.

Figure 11 Addresses of DSIM tables

```
ADDRESS OF CCW TABLE  =00012688
SIMD CCW ADDRESS TABLE =00012A58
SIMD CSW ADDRESS TABLE =00012AF8
TK1 RCAREA=00038743
TK0 RCAREA=00038348
ADDRESS OF TKTAB=0002FF76
DISPLAY TABLES IF NEEDED, HIT CR TO CONTINUE
```

Measurements

The creation or access of a data set on a direct-access device involves at least three specific activities at the device level: seeking (selecting the desired track on the device), searching (finding the desired record on the track), and reading or writing. The device characteristic tables identify timings appropriate to each of these functions. DSIM provides the measurement of the time required for the execution of these functions using the aforementioned timings. The channel and the control unit timings are not produced; however, provision has been made for this type of extension. In addition, DSIM reports the amount of space that was utilized, i.e., the number of tracks used, and the amount of data generated. Totals on the various commands issued to the device are also provided.

Concluding remarks

Functional simulation of direct-access storage devices allows for testing and experimenting with a current or new direct-access device without it being available or attached to the system. In particular, the MVE mode of DSIM allows the simulation of devices under any operating system. The techniques provided are data-driven, thus allowing the simulation of devices with differing characteristics. The interactive options enable the user to create or check for errors and thus provide for testing of the various error routines associated with the device. This capability is not generally attainable even with the real device. By extending the measurement routines, other device performance statistics may also be obtained. DSIM does not impair the security of the data within the system on which it is being used since it will not access any existing data within the system. DSIM is useful as an experimental intervirtual machine communication mechanism.

ACKNOWLEDGMENT

The author gratefully acknowledges the efforts of A. W. Hesse, G. W. Miller, S. B. Savage, G. H. Shoning, and Ms. J. M. Torre for providing programming support for the project and for contributing valuable suggestions.

CITED REFERENCES AND FOOTNOTE

1. D. N. Freeman, "Pseudo devices in OS/360," *Software Age* **2**, No. 5, 8,10,12-15,17,18,34-37 (June 1968).
2. A. J. Boutros, S. F. King, III, and J. D. Ruthledge, *Discrete Simulation of Direct Access Storage Device*, R. C. 3698, IBM Thomas J. Watson Research Center, Yorktown Heights, New York (January 1972).
3. E. Nahouraii and A. W. Hesse, "Simulation of virtual devices," *IBM Technical Disclosure Bulletin* **16**, No. 4, 1175–1176 (September 1973).
4. The simulator is an experimental development and is not available for distribution outside of IBM.
5. *IBM System/370 Principles of Operation*, Form No. GA22-7000, International Business Machines Corporation, Data Processing Division, White Plains, New York.
6. *IBM System/360 Operating System Input/Output Supervisor*, Form No. GY28-6616, International Business Machines Corporation, Data Processing Division, White Plains, New York.
7. *IBM Systems Reference Manual for IBM 3830 Storage Control and IBM 3330 Disk Storage*, Form No. GA26-1592, International Business Machines Corporation, Data Processing Division, White Plains, New York.
8. E. Nahouraii, *Direct-Access Storage-Device Simulation*, Technical Report TR 02.548, IBM, San Jose, California (March 7, 1973).
9. *IBM Virtual Machine Facility/370 (VM/370)*, Form No. GO20-1804, International Business Machines Corporation, Data Processing Division, White Plains, New York.