Published: 12/06/66

Identification

Overview of On-line Test and Diagnostic Operation
Harlow E. Frick

Purpose

The following parts of this section contain a general description of the on-line test and diagnostic system. The description for each system component is divided into integrated tests, periodic tests, and diagnostics in accordance with the philosophy described in section BR.0.01.

Integrated processor tests

The Multics supervisor checks for various unexpected conditions, including all unexpected processor faults. Unexpected conditions do not include any conditions which may be generated by undebugged ordinary user programs. They indicate a program error in the system software or a hardware failure.

Periodic processor test

The periodic processor test described in this section will be provided in the initial Multics T and D system. The final T and D system may replace or augment the test described here in order to correct some of the deficiencies; namely, no master mode or absolute mode tests, no instruction timing tests, and incomplete instruction sequence tests.

The purpose of the periodic processor test is to verify that a specific processor module is operating properly. It should not in any sense concern itself with operation of other system components except as required in order to execute the program.

Basic processor functions known to be operable because the Multics system is apparently operable, are not necessarily tested.

The processor time required to complete the test is expected to be quite small (probably less than 50 milliseconds). The period and priority level should be adjustable by the field engineer. The period of the processor test should be adjusted such that the test runs as often as possible without having a "noticeable" effect on system throughput. An execution frequency of somewhere between once per second and once per hour seems reasonable. The

reason for making the period as short as practical may be rationalized as follows. First, let's assume that the periodic processor test tests all of the processor logic. Also, let's suppose the system uses an average of 70 per cent of the processor logic over a 10 second period, 80 per cent over a 10 minute period and 90 per cent over a 1 day period.

Now, if the periodic processor test were never run a failure would go unreported until detected (and probably painfully isolated) by one or more users or an operator. Any number of users might be affected by the failure, but at least one would be.

If the periodic processor test were run once per day the failure would go unreported a maximum of one day and 10 per cent of the failures might be detected and corrected without user awareness (if an average of 90 per cent of the processor logic is used in one day).

If the periodic processor test were run every 10 minutes a failure would go unreported a maximum of 10 minutes and 20 per cent of the failures might be detected and corrected without user awareness (if an average of 80 per cent of the processor logic is used in 10 minutes).

If the periodic processor test were run every 10 seconds a failure would go unreported a maximum of 10 seconds and 30 per cent of the failures might be detected and corrected without user awareness (if an average of 70 per cent of the processor logic is used in 10 seconds).

Processor diagnostics

Processor diagnostics fall into the following categories:

1. A diagnostic run in the Multics environment on a processor which is currently being used by Multics. This diagnostic might be run under the following circumstances.

   a. Could be used as a periodic test.

   b. Could be executed automatically under control of Multics when an undefined system failure occurs.

   This type of diagnostic should be useful in diagnosing failures in processor logic which is not used by the Multics Supervisor or the file system and should be provided in the initial Multics T and D system.

2. A diagnostic run in the Multics environment on a processor which is not currently being used by Multics. Obviously, this type of diagnostic would apply only to multiple processor installations and could diagnose only certain categories of failures (admittedly a larger category than 1., above, but excluding absolute mode tests, timing tests, associative memory tests, and tests of the appending and paging hardware). One important point to note is that there could be a failure in the processor being diagnosed which might cause the system to crash even though the system were not using the processor except to execute the diagnostic. This type of diagnostic should be investigated as part of the feasibility study to determine requirements of the final T and D system.

3. Off-line diagnostics

   These diagnostics can perform a complete processor checkout. They do not run in the Multics environment and they require an entire system (GIOC with console, printer and magnetic tape, and one system controller) in order to operate.

Integrated system controller tests

Core parity errors will be detected by system software. The absolute location and data read should be available to the field engineer. When a core parity error occurs, the system software should abandon the 64 word page which the parity error occurred in and make the entire 64 word page available to the test and diagnostic system.

Periodic tests and diagnostics for system controllers

On-line tests and diagnostics for system controllers might be quite limited in the initial Multics T and D system. However, a feasibility study should be made to determine tests and diagnostics for system controllers which should be included in the final T and D system. (Note: System clock testing is considered separately in the last part of this section.)

Integrated tests for GIOC controllers

The major portion of the GIOC should be tested with integrated tests. The GIM (GIOC Interface Module) should include tests for correct GIOC hardware operation wherever possible. This approach is desirable because, initially it will help isolate software as well as hardware errors and because once the software is debugged, it will help isolate hardware malfunctions as GIOC failures rather than undefined failures.

Periodic tests for GIOC controllers

Periodic tests of the common GIOC logic would not be very useful because most failures would be immediately detected by the system or user. (Note: Common GIOC logic includes all GIOC logic except adapter channel logic) Thus, if the GIOC appears to function properly while being used by the system at a reasonable capacity, the probability of a failure being detected by a periodic test would be very low. Besides, periodic GIOC tests would be difficult to implement. Two approaches are possible:

1. Test the GIOC as it is being used by the GIM. This approach seems impractical even if one were to take unlimited freedom in the modifications made to the GIM to allow it to interface with the test program.

2. Periodically de-allocate the GIOC to customer use while it is tested. This approach would certainly work but it is doubtful if it would be worth the effort.

However, as part of the study to determine programs which should be provided for the final T and D system, this subject should be reviewed. It might be worthwhile (but I doubt it) to de-allocate an entire GIOC during slack time and test for certain GIOC failures like failures in priority logic which might not normally be detected by the system software. Another approach which should be investigated is the use of diagnostic GIOC commands to periodically test the GIOC while it is being used by the GIOC Interface Module.

GIOC diagnostics

No on-line GIOC diagnostics should be included in the initial Multics T and D system. However, a feasibility study should be made to determine GIOC diagnostics which should be included in the final T and D system.

Integrated tests for I/O devices

Generally, system software should check for correct operation of all I/O device commands and generate a file describing errors. A process should exist which analyzes and generates reports from the error file.

Periodic tests for I/O devices

Periodic tests will be provided for most or all of the system I/O devices because integrated tests cannot detect all I/O device failures and degradation on a timely basis.

All of the I/O device tests described below have priority and execution periods which may be individually adjusted for each device by the field engineer.

The Equipment Test Scheduler is a Multics module which schedules test programs to periodically run. This is done by making requests to the transactor at appropriate times and with appropriate priority. For example, the standard period might be adjusted such that each I/O device is tested no more than every six hours, and at least every 24 hours. Each of the tests described below should include a complete logical test of the device, excluding timing tests and tests which would require manual intervention after the test started.

1. Magnetic Tape Test

   The magnetic tape test might be integrated with maintenance of a library of scratch tapes. A user may wish to release a tape reel because he no longer needs it or because he suspects the tape may be damaged. He might do this by making the reel number available to the tape test.

   The Magnetic tape test would perform the following functions:

   a. Check handler for proper logical operation.

   b. Check handler for read write errors using known good tape.

   c. Check each tape reel which was previously made available by other user. The test would first make a write pass guaranteed to erase all private information on the tape. The test would then re-classify the tape as good or bad, and issue instructions to the operator as to the disposition of the tape.

2. Printer Test

   The printer test might consist of printing four printer pages. The first page might consist of an identification heading followed by an alignment test pattern. The second two pages might consist of a rotating pattern which causes each character to be printed in each column position. The fourth page might test those control characters and print commands which were not tested in the first three pages.

3. Card Punch Test

The card punch test might consist of punching a deck of about 200 cards. The front portion of the deck could contain identifier information. If the deck were successfully punched, instructions might be transmitted to the operator requesting him to save the deck. The next time a card reader test was scheduled, the operator could be requested to place all accumulated outputs from punch tests behind two known good decks and insert all in the card reader to be tested.

4. Card Reader Test

The card reader test might consist of reading and comparing to standard data, two known good decks of about 200 cards.

The test could also read and compare each deck punched by a punch test which had not yet been run through a card reader to verify that it was properly punched.

Diagnostic for I/O devices

At least two types of diagnostics will be available for each I/O device. One type will be provided by allowing the field engineer to specify his own diagnostic by inputting, compiling, and executing a program written in TDL. The other diagnostic will be a TDL diagnostic existing as a permanent file. This diagnostic might be similar or identical to the periodic test for that device.

Integrated tests for EMM's (Extended Memory Modules)

EMM logic should be tested with the standard EMM interface module, as the logic is used, whenever it is practical to do so. The following are types of features which it would be desirable to incorporate in the standard EMM interface modules.

1. Test that an Abnormal Status Word has been stored for each increment of the Status Entry Pointer.

2. Report all EMM errors to the field engineer.

3. Upon detection of a non-recoverable data error, effectively remove the EMM sector address from the system EMM memory map and add it to the diagnostic EMM memory map. (The diagnostic EMM memory map contains EMM sectors which the diagnostic system may access.)

4. Upon detection of a core parity error during a data transfer between core and EMM provide a facility for the field engineer to determine the core address of the parity error and the data read from that address.

Periodic EMM tests

Periodic EMM tests will not be included in the initial Multics T and D system because the need for a periodic test of EMM hardware, which is continuously being used, is very doubtful. However, routines to periodically check EMM error counters may be included.

EMM diagnostics

Complete EMM diagnostics will not be included as part of the initial Multics T and D System. A feasibility study should be made to define EMM diagnostics which should be included in the final T and D system. However, diagnostics for failures which are EMM address sensitive will be included as part of the initial Multics T and D system. The field engineer should be able to specify DCW's to be executed for any EMM address which is in the diagnostic EMM memory map. He should also be able to specify the data transferred and should receive messages reporting command results. These facilities should be provided by a new version of TDL. (TDL is described in section BR.0.01.)

At EMM initialization time there should be a facility for effectively transferring EMM sectors from the system EMM memory map to the diagnostic EMM memory map. The field engineer should be able to transfer sectors from the diagnostic EMM memory map to the system EMM memory map at any time. The reason to this facility is because each un-recoverable read error on the EMM is expected to transfer a page to the diagnostic memory map and it is likely that sometimes there will be a need to return these pages to the system EMM map without going through an initialization procedure.

Integrated system clock tests

Most system clock failures will cause an interrupt to notify the system of the failure. The system should respond to this interrupt with an appropriate error message.

Although the system clock is expected to be very reliable (MTBF of at least 8000 hours), and although most failures will be detected by an interrupt, there is a possibility of failures causing the calendar clock to count incorrectly

with no indication to the system. The probability of losing counts or reading all zeroes is higher than the probability of picking up counts. The system should therefore make periodic checks to insure that successive clock values are greater than the value previously read. It is not necessary that this be done by every user each time he reads the calendar clock because of the overhead required. However, it is recommended that this check be made in at least one system procedure which is used repeatedly and often. The routine which generates unique identifiers and process exchange are two recommended candidates.

Periodic system clock test

This section describes a testing scheme for system clocks which might be implemented in the initial Multics system. The test is outlined in some detail as an example which will help define details and potential problems in the Multics/diagnostic system interface.

If the system has two system clocks, both clocks should be set as accurately as possible and as close together as possible before Multics is initialized. During initialization, the system reads both clocks and uses the one with the slowest time as the primary clock. (Note: The slowest clock should be used as the primary clock because if a switch to the backup clock is necessary and if the backup block is behind the primary clock, the system must be locked up until the backup clock catches up to the last stored value of the primary clock. Otherwise there would be a possibility of generating duplicate unique identifiers.) If the clocks are within 5 seconds of each other, the clock with the fastest time is assigned as the backup clock and the system will begin operation in dual clock mode. If the clocks are not within 5 seconds of each other a message will be outputted indicating that only the clock with the slowest time is being used, and the system will begin operation in single clock mode.

If a clock failure is detected while in the single clock mode, the system is down until repair is made.

If a clock failure is detected while in dual clock mode, the system will continue operating in single clock mode with no user disruption.

The periodic system clock test will run only on systems currently operating in dual clock mode.

The purpose of the test is to:

1. Verify that both clocks are functioning properly by comparing the operation of the two clocks.

2. Switch to the good clock without user disruption when a failure occurs in either clock.

3. Generate messages describing failures and request adjustment when times drift out of prescribed limits.

The periodic system clock test will be initially awakened by the T and D demon if the system is initialized in dual clock mode. This might be done as follows:

1. Reset the primary wakeup flag.

2. Set the backup wakeup flag.

3. Store current times from both calendar clocks. (Note - This must be done by two adjacent store calendar clock instructions with bit 28 on in Master Mode. Thus, a facility must be provided to allow the T and D demon to do this.)

4. Request the primary clock test procedure to wakeup at 1/2 delta + current primary calendar clock time. (Note - Delta is the period of execution of the clock test.)

5. Request the backup clock test procedure to wakeup at delta + current backup calendar clock time via wakeup using the backup alarm clock. (Note - This means that the alarm clock co-ordinator must be able to perform scheduling using the backup clock for this procedure. Also, the procedure must be awakened only via an interrupt from the backup clock.)

When the primary clock test procedure is awakened, it might proceed as follows:

1. If the backup wakeup flag is on, go to Step 2. If the backup wakeup flag is off, it indicates either that we got a wakeup too soon from the primary clock or we have missed a wakeup from the backup clock. The following procedure might be executed to determine which of these error conditions occurred:

   a. Enter Master Mode and set bit 28 on to inhibit all interrupts.

   b. Save interrupt masks from each system controller.

   c. If interrupts were inhibited on either alarm clock, generate appropriate arguments and go to an "undefined error" procedure.

d. Set interrupt masks to mask all interrupts except alarm clock interrupt from the primary clock.

e. Set the system interrupt handling mechanism to immediately transfer control to the test procedure whenever a primary or backup alarm clock interrupt occurs.

f. Set bit 28 off to allow primary alarm clock interrupts.

g. If the primary alarm clock can be made to ring when the alarm is maintained greater than the current time, store the test case parameters and set a flag indicating that the primary calendar clock has failed.

h. Set interrupt masks to mask all interrupts except alarm clock interrupt from the backup clock.

i. If the backup clock can be made to not ring when it should, store the test case parameters and set a flag indicating that the backup clock has failed.

j. If neither or both clocks failed, generate appropriate arguments and go to an "undefined error" procedure.

k. If only one clock failed the test, generate an error message, switch the good clock as the primary clock, change to single clock mode, set the alarm clock to go off immediately, return the interrupt masks to their value at entry to the test, and exit. It should be noted that if a switch to the backup clock is necessary, and if the backup clock is behind the primary clock, the system must be locked up until the backup clock catches up to the last stored value of the primary clock. Otherwise there would be a possibility of generating duplicate unique identifiers.

2. If the primary wakeup flag is off, go to Step 3. If the primary wakeup flag is on, it indicates that we got more than one wakeup from the primary clock or that we missed a wakeup on the backup clock. A procedure similar to the one described in Step 1 above might be executed to determine which of these error conditions occurred.

3. Reset the backup wakeup flag.

4. Set the primary wakeup flag.

5. Store current calendar clock times in primary time and backup time.

6. Schedule the primary clock test to wakeup at primary time + delta.

7. Block this process.

When the backup clock test procedure is awakened, it might proceed as follows:

1. If the primary wakeup flag is on go to Step 2. If the primary wakeup flag is off, it indicates either that we got a wakeup too soon from the backup clock or we have missed a wakeup from the primary clock. An error routine could attempt to determine which of these error conditions occurred.

2. If the backup wakeup flag is off, go to Step 3. If the backup wakeup flag is on, it indicates that we got more than one wakeup from the backup clock or that we missed a wakeup on the primary clock. An error routine could attempt to determine which of these error conditions occurred.

3. If the difference between the primary and backup clock times is less than 5 seconds, go to Step 4.

   If the difference is between 5 and 30 seconds generate and output a message requesting the clocks to be adjusted (unless the message has already been given within the last 15 minutes) and go to Step 4.

   If the difference is greater than 30 seconds, mask interrupts and attempt to determine which clock is counting at the wrong rate. If this cannot be determined, output a message containing both times to the operator and request him to select which time should be used. After his response, set the clock he selected as the primary clock, switch to single clock mode, generate an error message to the field engineer and block this process.

4. Reset the primary wakeup flag.

5. Set the backup wakeup flag.

6. Schedule the backup clock test to wakeup at 1/2 delta + backup time (which was stored during the primary clock test procedure) via wakeup using the backup calendar clock.

7. Block this process.

System clock diagnostics

No on-line diagnostics are planned for the system clock. The system clock is designed such that almost any failure can be diagnosed, repaired, and tested off-line.